Authors: Kristy Fan, Edrei Chua, Josh Lange
Dartmouth College COSC 50
Group Project: Amazing Maze
August 24, 2015

Maze Systems Requirements Specification

DESIGN SPECIFICATION

In the Design Specification, we describe the input, data flow, and output specification for the Maze program.

(1) Input

The user calls the entire program to solve the maze by calling AMStartup in the terminal. AMStartup takes three parameters, each of which are optional. The three bracketed portions can be in any order. For each parameter the user enters, the user must specify which parameter the information is used for by denoting a hyphen and a letter before the parameter. The user should enter a command of the following form:

./AMStartup [-h (HOSTNAME)] [-d (DIFFICULTY LEVEL)] [-n (NUMAVATARS)]

Example command input: ./AMStartup -d 5 -n 2
./AMStartup
./AMStartup -n 8 -d 6 -h pierce.cs.dartmouth.edu

(HOSTNAME)

Description: The name of the server. In the scope of this assignment, the hostname, if specified, must be pierce.cs.dartmouth.edu.
Requirement: The user must have a working internet connection on a computer that can connect to the host. This parameter does not have to be entered. If the user does not specify a hostname, the program will default the hostname to **pierce.cs.dartmouth.edu**.
Usage: The program uses the hostname to get the IP address, which is used for the threads to communicate with the server.

(DIFFICULTY LEVEL)

Description: An integer between 0 and 9 (inclusive) indicating the difficulty of the maze to be solved. 0 is an easy maze of dimension 5x5, and 9 is a difficult maze of dimension 100x100.

Requirement: This parameter does not have to be entered. If the user does not specify a difficulty level, the program will default the difficulty level to **0**.
Usage: The program uses the difficulty level to ask the server for the appropriate maze that the program will solve.

(NUMAVATARS)

Description: An integer between 2 and 10 (inclusive) indicating the number of avatars in the maze to be solved.
Requirement: This parameter does not have to be entered. If the user does not specify a number of avatars, the program will default the number of avatars to **2**.
Usage: The program uses the number of avatars to ask the server for that many avatars in the maze to be solved.

(2) Output

If the program encounters an error during its run, the program will print an error specific to the problem. For more information on Error Conditions, see the Section in this Document entitled "Summary of Error Conditions."
If no error is encountered, the program will print a visual of the maze outlining the border of the maze and where each avatar is. If more than one avatar is at the same point in the maze, then instead of printing the ID of the avatar at that spot, the program will print a "+".

(3) Data Flow

AMStartup will create an initial message to send to the server, specifying that the program has created a socket, determined its number of avatars and difficulty level, and is ready to communicate to the server. The server will respond with a confirmation message, specifying the port number for further communication, the maze width and the maze height. AMStartup creates a log file to write in the future and creates the threads for avatars. The threads each have a package of information, known as AvatarData. For information on what the package contains, see the next section, Data Structures. Then, in AmazingClient, threads set up their information and send a ready message to the server. The server and avatar threads keep communicating, the threads sending requests to move in a direction and the server responding, until there are too many moves made or the maze is solved. The threads communicate with each other by sharing a pointer to a maze map that keeps track of which of the four directions are walls for each spot in the maze. Each time an avatar thread tries moves, the server return updated positions for all the avatars. From this information, the program determines whether that direction was a wall. If the avatar is in a new location, than that direction was not a wall, and if the avatar is in the same position, then that direction is a wall. The threads keep track of this information

in the MazeMap struct (described in the next section, Data Structures). This MazeMap struct is accessible to all other avatars so that the same wall will not be tried more than once.

(4) Data Structures

Each point in the maze has a coordinate, an x and y component, that is represented in a structure **XYPos**. The program uses a 2D array of **MazePoint** structures to represent each point in the maze. Each point in the maze has four directions and a XYPos coordinate to keep track of where the point is in the maze. Each direction can be one of three values: 1 if the direction is known to be not a wall, 0 if the direction is known to be a wall, and -1 if the direction is unknown. This 2D array is part of a **MazeMap** struct that also contains the width and height of the maze. The **AvatarData** structure is a package of all the data that the avatar threads need. It contains the avatar's ID, the total number of avatars, the difficulty level, the IP address, the port number, the MazeMap struct, and the filename of the log to which the progress reports will be printed, and the array of XYPos avatar locations. Finally, the messages sent between the threads and the server are of a structure **AM_Message**, which has many types:

- AM_INIT (the first message that AMStartup sends to the server specifying the number of avatars and the difficulty level)
- AM_INIT_OK (the message the server sends specifying the maze port, the maze width, and the maze height)
- AM_INIT_FAILED (the message the server sends if the AM_INIT message was ill-defined or if there was another errr)
- AM_AVATAR_READY (the message each avatar thread in AmazingClient sends first to the server)
- AM_AVATAR_TURN (the message the server sends updating which avatar's turn it is and what the avatars' positions are now)
- AM_AVATAR_MOVE (the message that each avatar thread sends the server asking to move in a certain direction)
- AM_MAZE_SOLVED (the message the server sends if all the avatars are at the same position in the maze)
- AM_UNKNOWN_MSG_TYPE (the message the server sends if the client-sent message had no appropriate type defined)

(5) Pseudocode

The blue code occurs in AMStartup. The red code occurs in AmazingClient.

```
// check the command line arguments
// create a socket for the client and connect the client to the socket
// send AM_INIT to server
```

```
// receive AM_INIT_OK message from server
// create file to write to
// create threads
// create a socket for the thread and connect the thread to the socket
// send AM_AVATAR_READY message to server
// set the goal vertex for the avatars to be where avatar 0 is
// while forever
        // if it's this avatar's turn
                // if the avatar is at the goal, do not ask to move
                // otherwise, figure out how to move according to right hand rule
        // send AM_AVATAR_MOVE to server
        // receive message from server
        // if server message is AM_MAZE_SOLVED
                // if avatar 0
                        // write to log file that solved
                        // clean up
                // cleanup
        // if server message is AM_TOO_MANY_MOVES
                // if avatar 0
                        // write to log file that too many moves
                // cleanup
        // if server message is AM_AVATAR_TURN
                // update the position of the avatar
                // update the locations array of the avatars
        // if server message is an error
                // if avatar 0
                        // write to log file that error
                // cleanup
        // if it's this avatar's turn
                // update the log with the new move and positions
                // update the maze map structure for new wall information
                // update the maze map file with new information
                // print map to the console
        // get the new turn ID
```

SUMMARY OF ERROR CONDITIONS

The program will print out an error message when any of the following conditions are met:

- There is an error with the provided parameters. Specifically,
    - If the number of avatars is not between 2 and AM_MAX_AVATAR, inclusive, or if the the number of avatars is not given, then an error message will be printed stating that the number of avatars will default to 2.
    - If the difficulty level is not between 0 and AM_MAX_DIFFICULTY, inclusive, or if the difficulty level is not given, then an error message will be printed stating that the difficulty level will default to 0.
    - If the hostname is not given or is not valid, then the an error message will be printed stating that the hostname will default to HOSTNAME.
- There is an error creating the socket.
- There is an error connecting the client to the socket.
- There is an error receiving any message from the server.

TEST CASES AND EXPECTED RESULTS

Test: Memory Leaks
Input: valgrind --leak-check=yes AMStartup
Output:
==12344== HEAP SUMMARY:
==12344== in use at exit: 0 bytes in 0 blocks
==12344== total heap usage: 260 allocs, 260 frees, 56,552 bytes allocated
==12344==
==12344== All heap blocks were freed -- no leaks are possible

For more examples of runs and results, please see the test log file in the main directory.