Documentation

(1) **Input**

Command Input
./AM_Startup -n [Number of Avatars] –d [Difficulty] –h [Hostname]

Example command input

AM_Startup –n 5 –d 4 –h stowe.cs.dartmouth.edu

[Number of Avatars] 5
Requirement: The number of avatars must be between 1 and 10, inclusive.
Usage: The program will inform the user and quit if the number of avatars is not in range.

[Difficulty] 4
Requirement: The difficulty must be between 0 and , inclusive.
Usage: The program will inform the user and quit if the difficulty is not in range.

[Hostname] stowe.cs.dartmouth.edu
Requirement: The hostname must be valid. The maze should be able to send details for the creation of the maze (Maze height, Maze width, MazePort).
Usage: The program will inform the user if the URL is invalid or if a connection to the server could not be established. Otherwise, the program will run.

---

(2) **Output**

The program will create an ASCII maze showing the avatars (labeled with their avatar IDs), solving the maze. Depending on whether the maze is solved or not, the program will inform the user of the same.
As the program runs, a log file is created. The first line of the log file has the MazePort, AvatarID, and the date and time. The rest of the log file contains all the moves that the Avatars have made.

---

(3) **Data Flow**

1. When AM_Startup is run, it creates a client socket which connects to the server.

2. Once successfully connected, the client sends an AM_INIT message that contains the number of avatars and difficulty.
3. After the message has been sent, the server responds with an AM_INIT_OK message.
4. The Mazeport, Maze Height and Maze Width are extracted from this AM_INIT_OK message.
5. AM_Startup now creates a log file with the name AMAZING_$USER_nAvatars_Difficulty. The first line of this file contains the Mazeport, Avatar ID, date and time.
6. Next, AM_Startup initializes all the memory that needs to be shared among the threads by allocating the relevant data structures to the heap.
7. The program initializes the list of last moves, the list of walls.
8. Finally, AM_Startup, callocs memory for the threads. For each thread, the AM_Args struct is initialized and its members (avatarId, nAvatars, difficulty, mazePort, ipAddress,logfile,width,height,walls,lastMoves, visits) are given the appropriate values.
9. Then each thread is created using the pthread_create() function which takes the AM_Args struct and the new_amazing_client which actually interacts with the server and makes the avatars move.
10. The new_amazing_client sends the AM_AVATAR_READY via the MazePort.
11. While the message received from the server is not AM_MAZE_SOLVED or AM_TOO_MANY_MOVES or AM_SERVER_TIMEOUT:
    I. The client waits for AM_AVATAR_TURN from server
    II. Checks past moves and writes message about the move to the logfile.
    III. Then it draws the move using ASCII graphics through the draw() function.
    IV. The lastMoves list is updated.
    V. The avatar with the correct avatar ID makes the move and sends the AM_AVATAR_MOVE message. The move is made using the left hand traversal algorithm.
    VI. Once the maze is solved, one of the avatars writes that down.
12. Once the maze is solved, the logfile is closed, the memory allocated in the heap is freed.

---

(4) **Data Structures**

Walls- A 3D array that stores each maze-tile (the X,Y position) and direction (N,S,E,W).

Visits- A 3D array that is used to determine whether a maze tile has been visited. If visited, it is indicated using '1', otherwise '0'. It gives constant time access to visits.

Move – Stores the X and Y position of the avatar, the direction (if the move was successful) and the attempted direction (if the move was unsuccessful) of the avatar.

AM_Args – Data structure that is passed into each of the n threads. Stores the avatar ID, number of avatars, difficulty, IP Address, MazePort, width, height, walls array, lastMoves list, log file.

---

(4) **Pseudocode**

1. Check that the arguments are valid
    I. Correct number of arguments
    II. Valid number of avatars
    III. Valid difficulty
    IV. Valid option characters
2. Use gethostbyname() to acquire the server information.
3. Connect to the server using the 'sockfd' obtained from the socket() command and the information obtained from gethostbyname().
4. Construct the AM_INIT message, send it to the server and then receive the message from the server.
5. If the message received is not AM_INIT_OK, exit out and inform the user.
6. Otherwise, acquire the maze information: Maze Height, Maze Width and MazePort from the AM_INIT_OK message.
7. Convert the maze information to the Host Byte Order using ntohl() function.
8. Use getenv() to get the user ID, calculate the length of the log file name, calloc memory for the name and then use sprinf() to make the name.
9. Use curtime() to get the time of the function call.
10. Allocate memory for the structs that will be shared among the threads : walls, lastMoves.
11. In a for loop, that loops as many number of times as there are number of avatars, initialize a struct that contains all the relevant information that each thread needs and then pass that information along with the new_amazing_client through the pthread_create() function.
12. Once all the threads have been created, use pthread_join() for all the threads.
13. Each thread runs the new_amazing_client which does the following:
    I. First, perform argument checking and check for invalid pointers.
    II. Create a socket to connect to the server.

III. Construct and send the AM_AVATAR_READY message to the server.
IV. While the program has not received an IS_AM_ERROR or AM_MAZE_SOLVED message from the server, the avatar detects if it is its turn to move. The avatar checks if the previous avatar made a successful move. If the previous avatar ran into a wall, then the program will update the list of wall/ If ot, then the current avatar stores the previous avatr's attempt as its last move.
V. The avatar generates a move to send to the server.
VI. Once the maze has been solved (if it receives the AM_MAZE_SOLVED message from the server) or if there is an error (if it receives an IS_AM_ERROR message from the server), the program will close the log file and return NULL.

14. Close the logFile and free up heap allocated memory.