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.

1. **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.

1. **Data Flow**
2. When AM\_Startup is run, it creates a client socket which connects to the server.
3. Once successfully connected, the client sends an AM\_INIT message that contains the number of avatars and difficulty.
4. After the message has been sent, the server responds with an AM\_INIT\_OK message.
5. The Mazeport, Maze Height and Maze Width are extracted from this AM\_INIT\_OK message.
6. 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.
7. Next, AM\_Startup initializes all the memory that needs to be shared among the threads by allocating the relevant data structures to the heap.
8. The program initializes the list of last moves, the list of walls.
9. 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) are given the appropriate values.
10. 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.
11. The new\_amazing\_client sends the AM\_AVATAR\_READY via the MazePort.
12. While the message received from the server is not AM\_MAZE\_SOLVED or AM\_TOO\_MANY\_MOVES or AM\_SERVER\_TIMEOUT:
13. The client waits for AM\_AVATAR\_TURN from server
14. Checks past moves and writes message about the move to the logfile.
15. Then it draws the move using ASCII graphics through the draw() function.
16. The lastMoves list is updated.
17. 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.
18. Once the maze is solved, one of the avatars writes that down.
19. 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).

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, logfile.

1. **Pseudocode**
2. Check that the arguments are valid
3. Correct number of arguments
4. Valid number of avatars
5. Valid difficulty
6. Valid option characters
7. Use gethostbyname() to acquire the server information.
8. Connect to the server using the ‘sockfd’ obtained from the socket() command and the information obtained from gethostbyname().
9. Construct the AM\_INIT message, send it to the server and then receive the message from the server.
10. If the message received is not AM\_INIT\_OK, exit out and inform the user.
11. Otherwise, acquire the maze information: Maze Height, Maze Width and MazePort from the AM\_INIT\_OK message.
12. Convert the maze information to the Host Byte Order using ntohl() function.
13. 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.
14. Use curtime() to get the time of the function call.
15. Allocate memory for the structs that will be shared among the threads : walls, lastMoves.
16. 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.
17. Once all the threads have been created, use pthread\_join() for all the threads.
18. –AM\_Client—
19. Close the logFile and free up heap allocated memory.