AMStartup

Input: Number of avatars, Difficulty of Maze, Hostname.
Output: Error messages, Image of maze, logfile.
Data Flow:
AMStartup initializes the server and collects the basic information (dimensions, mazeport). It then initializes the logfile and *Leaders* and *Info* structs.

GUIinit builds an empty Maze struct of the correct size and places avatars at their starting points
Avatar takes the information from AMStartup. It runs a turn, receives the response, stores it in *Info*, and passes the current position of all avatars and the original position of the avatar whose turn it was to GUIupdate, which updates the shared *Cell_t Maze[x][y]* struct. *Info* is then updated with the new direction. This continues until an error or maze complete message is received.

Move takes the current positions of the avatars, the position of a single avatar, and the shared *Cell_t Maze[x][y]* struct. It checks for movement, then adds walls and paths to the shared *Cell_t Maze[x][y]* struct based on whether or not the new position matches the old one. Finally, it closes off dead ends in the maze and updates the walls in the *Cell_t Maze[x][y]* struct to reflect them.

Data Structures:
Cell_t: array of 4 ints for the walls followed by a pointer to store an array of ints tracking the number of times each avatar has visited the cell
Cell_t Maze[x][y]: double array of cells representing every space in the maze.
Leaders: array of ints describing whether a given avatar is following a path or laying its own.
Info: the memory of every individual avatar. Stored as an array, each structure holds the avatar ID, its socket, the maze port, the hostname, the last message from the server, and the last direction the avatar traveled in.
Pseudo Code:

```
1. Validate inputs.
2. Create socket and initialize the server address.
3. Connect socket to server and send initialization message. The dimensions and maze
   port are obtained from the AM_INIT_OK message.
4. Make the logfile title and open it for writing. Print the first line of the log.
5. Initialize n threads, where n is the number of avatars, and have each avatar send and
   AM_AVATAR_READY message to the server.
6. Call GUIinit to build the empty maze
    a. GUIinit takes a height and width and makes an empty Maze struct from it
    b. GUIinit then sets the outside wall values
    c. Finally, GUIinit displays the maze with the avatars in their cells
7. Until an AM_MAZE_SOLVED is recieved, loop over each thread:
    a. Check if it is that avatars turn
```

```
        b. If it is, call Avatar to determine move direction
        c. Send the move to the server
        d. Get the response
            i. If it's an error or AM_MAZE_SOLVED, break.
        e. Update the maze and the GUI
    8. Clean up
        a. Close all threads
        b. Free the Maze struct
    9. Report Success/Failure
```

Avatar (a.k.a alg)

Input: initialized AM_Message, Maze height, Maze width, Double array of cells, Avatar ID, number of avatars, Leaders array, last direction taken
Output: number between 1-4 corresponding to the chosen direction.
Data Flow:
Avatar takes an initialized AM_Message, chooses a direction and changes the Leaders array, and sets the AM_Message attributes based on it. The AM_Message is then sent by AMStartup, which sends the message and updates the Maze struct based on the result of the move.
Data Structures: Uses the Cell_t Maze[x][y] and Leaders, but no unique ones needed.
Pseudo Code:

```
    1. Send AM_AVATAR_READY
    2. For every message received, check if it's an error message. If yes, return a non-zero
       error code to end all Avatar threads.
    3. Likewise, check if AM_MAZE_SOLVED is received (technically possible on turn 0)
        a. If yes, return a 0 value
    4. Wait until AM_AVATAR_TURN with matching Avatar ID is received
    5. Start logic
        a. Check if the avatar is a leader:
            i. If yes, continue.
            ii. If no, skip to (e).
        b. Check if the avatar is the last leader left
            i. If no, continue.
            ii. If yes, check if it is in a dead end.
                1. If not, stop.
                2. If yes, move out of the dead end. This prevents an edge case
                   where the followers will leave and the leader won't
        c. Check if avatar is on a tile that has been explored by another avatar
            i. If yes, check if the avatars that explored it are all followers
                1. If they all are, continue
                2. If one of them is not, the avatar becomes a follower. Skip to (e).
```

   d. Check if there are any unexplored walls left
      i. If yes, try to go in that direction in this order: North, East, South, West
         1. (That is, if the North and West walls are closed, try to go East. If that is also filled, go South).
      ii. If no, make a left turn
   e. If the avatar is a follower, check if a leader has left the square (Cell_t map[currentx][currenty].walls[i] = 3)
      i. If yes, follow it in that direction.
      ii. If no, sleep.

6. Send the appropriate AM_AVATAR_MOVE message
7. Return the direction code