—Amazing Maze—

Design and Specifications

Espadeiro

I. Amazing Maze Overview

Amazing Maze by Espadeiro is a program designed to solve complete mazes by directing avatars along a path that will bring them all together. Avatars take turns moving and, using the information gathered through each turn, a path is determined leading all the avatars to a solution. Due to the nature of a complete maze, this path is also guaranteed to be ideal. Amazing Maze also supports functionality allowing a graphical representation of the maze to be constructed as the information becomes available to the local machine.

Amazing Maze will consist of a startup script and an Avatar controller program. The design specifications for both are given below.

II. Amazing Maze Startup Design Specifications (AMStartup)

Input:

1) –n nAvatars: the numbers of Avatars in the maze (int 2-10).
2) –d Difficulty: (int) the difficulty level, on a scale from 0-9
3) –h Hostname: (char*) the hostname of the server running the maze.

Output:

Amazing Maze Startup will generate Avatar Client threads for each of the avatars specified upon completion of the program.

Data Flow:

AMStartup will communicate with the server and receive information packets back. It will then process this information and use it to generate avatar client processes.

Data Structures:

Avatar—a struct to hold int and char* information received by MazePort for use in Avatar Client

AM_Message – contains message information.

Pseudo Code:

- Begin by checking arguments. If all arguments are valid, continue. Otherwise inform user of mistakes.
- Instruct network component to initiate contact with specified server (AM_INIT). Check for success (AM_INIT_OK).
- Initialize a thread for each avatar.

- Store the received information about MazePort and Avatars, and for each Avatar, initiate an Avatar Client.
- Exit when inferior process report an exit condition (see the Avatar Client)

III. Amazing Maze Avatar Client Design Specifications

Input:

1) AvatarId (Integer generated in Amazing Maze Startup starting at 0 and incrementing by 1 for each additional avatar).
2) The total number of avatars in the maze.
3) The difficulty of the maze.
4) The IP address of the server
5) The MazePort returned from AM_INIT_OK message.

Output:

Amazing Client will create the log file specified for it's avatar and update it with each move made. Amazing Client will also generate global structures after each turn acting as bread crumbs and informing the actions of the other avatars. As each avatar makes its move, it will save the current knowledge it has created about the total graph for future us by it and the other avatars.

Data Flow:

Information will be generated through the server-client interactions in the form of valid and invalid moves. This information will be stored within the shared memory Maze Struct so that avatars will know when they have crossed paths and will also be able to keep track of valid moves to avoid repeated experimentation. Additional information from the server will include the END type messages associated with success or some failure condition.

Data Structures:

Maze – Holds the MazeNode array as well as the maximum x and y coordinates. Has methods for traversal that allow the maze array to be used like a graph.

MazeNode** grid - Two-dimensional array which holds information about each square in the maze. Stored in shared memory accessible to each Avatar Client and updated after each move. Designed such that the MazeNode also functions like a graph for traversal. Additionally, can be accessed by graphics to draw the maze as information is generated.

MazeNode – Holds the "bread crumb" information for a given move. This includes information about walls that have been discovered and whether the node has been visited (and by which avatar). It also includes binary and number of moves values to implement BFS.

Avatar – a struct holding info for each of the avatars in the maze. This includes their x and y positions (XYPos).

XYPos – Holds integer values for x and y positions.

Tuple – Holds a tuple. Intended for x, y, and LastMoveDirection integers.

Myvalues – Holds useful information such as the number of avatars, the avatar ID, difficulty level, maze port, IP address, and the name of the logfile.

Pseudo Code:

- Process Parameters received from AMStartup
- Once ready, send AM_AVATAR_READY message via MazePort with AvatarId.
- Process AM_AVATAR_TURN message.
- Initialize data structures
    - Avatar's graph starts of with one node (current location) and with no known edges.
    - Inform global 2 dimensional array of current location. (X,Y as keys).
- If TurnID indicates that it is the current Avatar's move, the Avatar moves forward using the left hand rule and makes it's way around the maze. The idea is the discovery of the maze as much as possible and reduction by taking out the dead ends. Left Hand rule works as follows:
    - The avatar before every move checks for a move to go to it's left. If it can't it tries to go straight. Even if that is not allowed, it goes to it's right. Otherwise it goes back.
- So all the avatars on their turn keep discovering the maze till one of them which is Avatar 0 runs bfs and realizes that it can follow all other Avatar's trails to get to them and that they are well connected. This is done using bfs on the condition:
    - If a certain avatar has been to a node that Avatar records in that Node the information about the directions that the Avatar arrived and left that Node, which gives us allowed directions into and out of the certain Node in the Maze.
- Then Avatar runs bfs and basically finds the path using allowed directions to all other Avatars and gives them its own coordinates as the convergence point.
- Then the other Avatars with the information that they are somehow connected use bfs to get to Avatar 0.

IV. Amazing Maze Design Overview

Khizer, before we submit, make sure we have the final tree (if you change any files)

```
amstartup
├── bin
│   ├── amstartupclient2.0.o
│   ├── avatardaemon2.0.o
│   ├── maze-2.o
│   └── placeholder
├── genlogs.sh
├── graphics
│   ├── asciigraphics.c
│   ├── MazeSolved
│   │   ├── avatar_log.txt
│   │   ├── cs1lib.py
│   │   ├── cs1lib.pyc
│   │   ├── load_maze.py
│   │   ├── load_maze.pyc
│   │   ├── maze_draw.py
│   │   ├── maze.py
│   │   ├── maze.pyc
│   │   ├── mazetest.dat
│   │   └── test_maze.txt
│   └── placeholder
├── library
│   ├── queue.c
│   ├── queue.h
│   ├── stack.c
│   ├── stack.h
│   └── testscripts
│       ├── queuetest.c
│       ├── queuetest.log
│       ├── stacktest.c
│       └── stacktest.log
├── logfiles
│   ├── Amazing_khizarhussain16_2_2.log
│
├── Makefile
├── maze.dat
├── network
│   └── whatDoIhave
├── README.md
├── Spec.docx
├── Spec.pdf
├── src
│   ├── amazing-2.h
│   ├── amstartupclient2.0.c
│   ├── avatardaemon2.0.c
│   ├── avatardaemonold.c
│   ├── maze-2.c
│   └── maze-2.h
└── testing
    ├── EllaREADME
    ├── KhizarREADME
    └── mazetest.c
```

The overall program will be implemented in self-contained modules relying on shared data structures and source code. By disciplined use of consistent interfaces, this will allow for modules (graphics, controller, logic, network) to be modified without requiring the restructuring of the entire program.