Authors: Gabriel Corso, Tyler Crowe, Joanne Zhao
CS 50 - Spring 2014
Assignment: The Amazing Project

Amazing Project Design Spec

A client application (avatar) will simulate avatars searching for each other in a virtual maze generated and maintained by a separate server application. The AMStartup module opens up a connection to the server and initializes the avatars. The avatars will send messages to the server with attempted moves, and the server will return messages depending on the success of such moves. The server will communicate to the avatars when the maze has been solved – when all the avatars are at the same location in the maze. The progress of the maze solving will be logged in a file.

AMStartup Design Spec

In the following Design Module we describe the input, data flow, and output specification for the amazing project. The pseudo code for the amazing project is also given.

The Design Spec includes:
(1) Input: Any inputs to the module
(2) Output: Any outputs of the module
(3) Data Flow: Any data flow through the module
(4) Data Structures: Major data structures used by the module
(5) Pseudo Code: Pseudo code description of the module.

(1) Input

Command Input

Running the amazing project:

AMStartup [NUMBER_AVATARS] [DIFFICULTY] [HOSTNAME]

example:

AMStartup 5 4 pierce.cs.dartmouth.edu

[NUMBER_AVATARS]

Number of avatars to place in the maze.

Requirements: Must be an integer greater than or equal to 2, less than or equal to 10.

[DIFFICULTY]

The difficulty of the maze, determines the max number of moves and the size of the maze

Requirements: Must be an integer between 0 and 9 inclusive.

[HOSTNAME]

The hostname of the maze server

Requirement: Must be able to establish a connection

(2) Output

Alert messages are displayed to console when messages are sent to and from the server. A log file entitled “AMAZING_$USER_num_diff.log” (where $USER is the username, num is the number of Avatars, and diff is the difficulty of the maze) is generated. The first line of the file will contain the $USER, the MazePort, and the date & time.

The maze port, maze width and maze height, which will be returned by the server via an AM_INIT_OK message, will also be printed to the console.

(3) Data Flow

AMStartup will check all the arguments. Then, it will initialize a connection with the server, sending an AM_INIT message. The server is expected to respond with an AM_INIT_OK message. AMStartup will extract the maze port, maze width and maze height, which will be included in this message.

The log file, “AMAZING_$USER_num_diff.log” will be created.

Then, AMStartup will start n number of forked processes (where n is the number of avatars), each with the AvatarId (an integer generated by AMStartup, starting at 0 and incremented by one for each subsequent Avatar started), the nAvatars (total number of Avatars), the Difficulty (difficulty of the maze), the IP address of the server, the MazePort (as returned in the AM_INIT_OK message) and the Filename of the log the Avatar should open for writing in append mode.

4) Data Structures and Major Variables

---

nAvatars - the number of avatars for the maze

difficulty - the difficulty of the maze
IPAddress - the IP address
fileName - the log's file name
mazePort - the TCP/IP Port number that should be used to communicate with the server about this new maze
mazeWidth - the width of the maze
mazeHeight - the height of the maze

The following messages will be used to communicate between AMStartup and the server

AM_INIT - AMStartup builds the AM_INIT message with the **nAvatars** and **difficulty** and sends it to the server on the AM_SERVER_PORT port identified in amazing.h.
If the initialization succeeds, the server will respond with an AM_INIT_OK message.
If nAvatars is greater than AM_MAX_AVATAR (10) or the Difficulty is greater than AM_MAX_DIFFICULTY (9), the server will respond with an AM_INIT_FAILED message.

AM_INIT_OK - If an AM_INIT message is successfully processed by the server, the server will respond with this message containing a TCP/IP port number that Avatars should use to communicate with the server.

AM_INIT_FAILED - If an AM_INIT message is NOT successfully processed by the server, the server will respond with this message containing an error code. For the possible values and their meaning, see the amazing.h file.

(5) AMStartup Pseudocode

---

1. Check the arguments, getting the number of avatars, difficulty, hostname
2. Get the IP Address from the host name
3. Set up the log file
4. Create a socket
5. Connect to the socket
6. Create the AM_INIT signal with the number of avatars and difficulty
7. Send the message to the server, if unsuccessful terminate
8. Receive a message from the server. If AM_INIT_OK, continue
9. Write first line of the file with $USER, mazeport and date.
10. Start N processes with Mazeport, MazeWidth, MazeHeight.

Avatar Design Spec

In the following Design Module we describe the input, data flow, and output specification for the avatar module. The pseudo code for the avatar module is also given.

The Design Spec includes:

(1) Input: Any inputs to the module

(2) Output: Any outputs of the module

(3) Data Flow: Any data flow through the module

(4) Data Structures: Major data structures used by the module

(5) Pseudo Code: Pseudo code description of the module.

(1) Input

Running the avatar module after the maze has been generated

./avatar [AvatarId] [nAvatars] [Difficulty] [IPAddress] [MAZEPort] [LogFile]

Example:

./avatar 0 3 2 129.170.212.235 10829 Amazing_3_2.log

[AvatarId]

An integer generated by AMStartup, starting at 0 and incremented by 1 for each subsequent Avatar generated

[nAvatars]

An integer greater than or equal to 2 and less than or equal to 10. This represents how many Avatars total in the maze

[Difficulty]

An integer greater than or equal to 0 and less than or equal to 9. This number determines the size of the maze and the max number of moves allowed.

[IPAddress]

This is the IP Address of the server.

[MazePort]

This value is returned from AM_INIT_OK and is the port number for the generated maze.

[LogFile]

Filename of the log the Avatar should open for writing in *append mode*. In the format AMAZING_<USERNAME>_<nAvatars>_<Difficulty>.log

(2) Output

The avatar program writes information to both the log file and the console while running. The log file will include the moves made by the avatars. When the maze is completed, the log file will include a line by one of the avatars confirming that the maze has been solved. A graphical visualization is also displayed using ASCII.

(3) Data Flow

Once the AMStartup has been run the avatar sends an AM_AVATAR_READY message to the server via the MazePort attached is the AvatarId. The avatar will initialize the maze data structure to the size of the maze on the server. First avatar calculates who the master avatar will be (the avatar with the smallest distance between every avatar). The server responds with the avatars current position in the maze and a TurnID to indicate the order of moving. These will be stored in shared memory for all the avatar processes to access.

The avatar will send a AM_AVATAR_MOVE message to the server to specify where it will try to move. If it can move its position is updated. The server sends back an AM_AVATAR_TURN message to all the avatars with the position of all the avatars including the avatar that just tried to move. The server increments TurnID so the next avatar can move.

When the server sends back the AM_AVATAR_TURN message, the avatar that just tried to move will update his map. If he successfully moved, increment the value of the node he was just at in the map, otherwise set the wall in the direction he tried to move to true. Update the master map with the wall.

Continue this until one avatar finds the master avatar. His path will be added to the mastermap so if any other avatar lands on the path the avatar will just follow the same path to the master avatar without checking the other paths. As each avatar finds the master their path will be added to the master map for the other avatars to follow to find the master.

Once all the avatars find the master, finish.

4) Data Structures

XYPos - a simple data structure that holds x and y coordinates

Avatar - holds XYpos, Avatar ID, leader/follower

Maze - an array of arrays to match the size of the map. Each entry corresponds to a square in the map. At each entry there will be a pointer to a maze node structure.

Maze Node - Each maze node will hold boolean variables for the walls on each edge of the maze square. The walls will be called NORTH, SOUTH, EAST and WEST corresponding to a north wall, south wall, east wall, or west wall at that square. It will also hold a variable to count the amount of times it has been walked over.

Each avatar will hold an instance of the maze, and update it as they journey through.

The following messages are sent between the avatar and the server:

AM_AVATAR_READY - When ready to begin moving through the maze, each of the Avatars send the server a ready message with its assigned AvatarId. If the AvatarId is valid, the server will respond with an AM_AVATAR_TURN message after all of the Avatars are ready. If the AvatarId is invalid, the server will respond with an AM_NO_SUCH_AVATAR message and continue waiting for a valid ready message.

AM_AVATAR_MOVE - To attempt a move, an Avatar sends a move message to the server with their AvatarId, and an integer indicating which direction (M_WEST, M_NORTH, M_SOUTH, M_EAST) the Avatar wishes to try to move (defined in amazing.h). It is also possible for the Avatar to indicate to the server that no move is desired (M_NULL_MOVE).

The server will validate the AvatarId for this maze, and respond with AM_NO_SUCH_AVATAR if it is invalid. Otherwise, the server proceeds to determine whether or not that move is possible. For example, the Avatar cannot move outside of the maze or through a wall. If the move is valid that Avatar's position in the maze will be updated.

AM_NO_SUCH_AVATAR - This message is sent by the server in response to any message from an unknown AvatarId.

AM_AVATAR_TURN - Before the first move, the server waits until it has received AM_AVATAR_READY messages from all N Avatars. Once that occurs, the server responds to each of them with an identical AM_AVATAR_TURN message.

The Avatars take turns making a move in the maze. After each Avatar move is received and processed, the server sends an updated version of this message to all of the Avatars. This message will have the moving Avatar's possibly updated (x,y) position and the TurnID incremented (modulo N) to enable the next Avatar to move.

The message includes the TurnID (the AvatarId of the Avatar whose turn it is to move next), followed by (x,y) pairs representing the current positions of each of the N Avatars: Avatar0, Avatar1, … , Avatarn-1.

AM_MAZE_SOLVED - When an Avatar sends a move request to the server that results in all of the Avatars being located at the same (x,y) position, the server will respond with a special message containing a proof that you solved the maze. When the Avatars receive this message, one of them should ensure that the message is written to the log file. (Hint: use printf, fprintf, or sprintf after using ntohl()). These values and the accompanying hash are proof that you succeeded. Once an Avatar receives this message, the Avatar should clean up (free memory, release shared resources, close sockets, close files, etc.) and exit.

AM_UNKNOWN_MSG_TYPE - If the server receives a message that it can't understand, it will respond with an error message, and the unrecognized message type.

AM_UNEXPECTED_MSG_TYPE - If the server receives a message type out of order, this message will be returned. If this error is encountered during initialization, the server will abort and die.

AM_AVATAR_OUT_OF_TURN - If an Avatar sends an AM_AVATAR_MOVE message to the server when it is not that Avatar's turn to move, the server will ignore that move request and respond with this message.

AM_AVATAR_TOO_MANY_MOVES - If an Avatar sends an AM_AVATAR_MOVE message to the server and that move exceeds the maximum number of moves for a given maze (some function of AM_MAX_MOVES and Difficulty), the server will return this message to all Avatars. The maximum number of moves (all avatar's moves added together) is: AM_MAX_MOVES * (Difficulty+1) * nAvatars. For example, at Difficulty 6 and 4 Avatars the maximum number of moves would be 28,000. Each Avatar should then cleanup and exit. This limit is in place to guard against endless loops in the Avatar code.

AM_SERVER_TIMEOUT - If at any time the server waits for AM_WAIT_TIME seconds without receiving a message from a client, the server will send this message and abort the maze.

AM_SERVER_DISK_QUOTA - The server produces many files associated with a given maze. These files can take up quite a bit of disk space, especialy with many clients solving higher difficulties. If the server cannot create or write to a file because of a disk quota error, this message will be sent.

AM_SERVER_OUT_OF_MEM - In the rare case that the server cannot allocate enough memory to serve a maze, this message will be sent. The server will then abort and die.

(5) Avatar Pseudocode

---

1. Initialize maze to size returned by the server
2. Calculate the master avatar
3. while the avatar has not found the master and has not reached the max number of movesπ
4. try to move towards the master. If the avatar is the master try to move to a T intersection
5. if the avatar could move
6. increment the trail at the node in the map we were just at
7. if the avatar could not move
8. update the wall on his instance map and the shared maze
9. update log file with turn
10. Master found travel with master
11. If last avatar to find master
12 update finish to log file

** The shared memory maze is used for wall information

** The individual mazes are used for trail information