Amazing Project Design Spec

Team Joubertin
Hao Chang, Hanna Kim, Matt Krantz

Describes the input, data flow, and output specifications for the main modules of the amazing project (AMStartup and Avatar). The pseudo-code for these modules is also given.

The design spec consists of two key pieces:
(1) The AMStartup program
(2) The Avatar program

Each piece of 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

AMStartup Program

(1) Input

Command Input:

```
$ AMStartup -n [NUM AVATARS] -d [DIFFICULTY] -h [HOST NAME]
```

Example Command Input:

```
$ AMStartup -n 4 -d 3 -h tahoe.cs.dartmouth.edu
```

[NUM AVATARS] 4
Requirement: Must be a positive integer
Usage: AMStartup needs to inform the user if [NUM AVATARS] is invalid. If it is valid, the program creates a separate client [NUM AVATARS] times

[DIFFICULTY] 3
Requirement: Must be an integer between 0 (very easy) and 9 (extremely difficult)
Usage: AMStartup needs to inform the user if [DIFFICULTY] is not an integer, or if it does not fall within the required range. If valid, integer used to determine difficulty of maze

[HOST NAME] tahoe.cs.dartmouth.edu
Requirement: The hostname of the server to run program on (for final, pierce.cs.dartmouth.edu) - must be a valid server name

Usage: The program must inform the user if the server is invalid or unavailable (does not respond with AM_INIT_OK)

(2) Output

AMStartup produces a log file with the name Amazing_$USER_N_D.log, where $USER is the current user id, N is the value of [NUM AVATARS] and D is the value of [DIFFICULTY]. The first line of the file contains $USER, the MazePort, and the date and time. As the program runs, each avatar updates the log file created by AMStartup in turn (see Avatar spec below). AMStartup also prints details about the maze it is going to solve to stdout.

(3) Data Flow

After validating its arguments, AMStartup constructs and sends the AM_Init message to the server. When the server responds with AM_INIT_OK, AMStartup recovers the MazePort from that reply. The program then creates the log file (described above) and starts [NUM AVATARS] processes running the main client software. Each process is given the appropriate start parameters. If the program encounters an error, or all avatars exit, it will terminate.

(4) Data Structures

While AMStartup does not make use of any data structures, it stores and uses the following information:

avatarID - starts at 0, incremented for every avatar created by AMStartup

numAvatars - stores number of avatars

diff - stores the difficulty

mazePort - stores the maze port, as given in AM_INIT_OK

width - stores width of maze
- Along with height, helps create grid for each avatar
- Included in AM_INIT_OK message

height - stores height of maze
- Along with width, helps create grid for each avatar
- Also included in AM_INIT_OK message

logFile - name of log file

- Each avatar opens for writing in append mode

(5) AMStartup Pseudo-code

```
Server is running, waiting for clients

Check command line arguments
// Inform user if invalid and exit

Generate AM_INIT message

Send AM_INIT to server

If server does not respond with AM_INIT_OK:

     Something went wrong
     // Exit

Store information from AM_INIT_OK
// Height, width, mazePort, position, etc.

For 0 through numAvatars - 1:

     Start a new avatar process with appropriate params
     // Determined from stored information

When all avatars meet at the same cell or exit:

     Display relevant message
     // Success or error

Exit
```

Avatar Program

(1) Input

Command Input:

```
$ avatar [ID] [NUM AVATARS] [DIFFICULTY] [IP ADDRESS] [MAZE
PORT] [LOG FILE] [MAZE WIDTH] [MAZE HEIGHT]
```

Example Command Input:

```
$ avatar 0 3 2 10.31.192.213 10829 Amazing_mlkrantz_3_2.log 12
12
```

* Note: Because the client (avatar) program is not meant to be run by people, the start parameters are positional and required

[ID] 0
Requirement: Must be passed to the program from AMStartup (should be between 0 and number of avatars - 1)
Usage: Used to identify avatars for traces and other purposes

[NUM AVATARS] 3
Requirement: Same as for AMStartup program
Usage: Used to determine size of x-y coordinate arrays

[DIFFICULTY] 2
Requirement: Passed to program from AMStartup
Usage: Determines maze difficulty (see requirements and usage in AMStartup spec above)

[IP ADDRESS] 10.31.192.213
Requirement: The IP address of the server - must be a valid, reachable IP address
Usage: The program will exit if it cannot connect; otherwise, used to listen for messages from server

[MAZE PORT] 10829
Requirement: Must be valid maze port, as passed to program from AMStartup (obtained in AM_INIT_OK)
Usage: The program will exit if it cannot connect; otherwise, used to listen for messages from server

[LOG FILE] Amazing_mlkrantz_3_2.log
Requirement: Must be a valid log file created by AMStartup
Usage: Opened by each avatar in append mode, used to log progress as avatars take turns and move

[MAZE WIDTH] 12
Requirement: Must be passed to the program from AMStartup, corresponds to the difficulty level
Usage: Used to specify the dimensions of the grid (array of cells stored by each avatar)

[MAZE HEIGHT] 12
Requirement: Must be passed to the program from AMStartup, corresponds to the difficulty level
Usage: Used to specify the dimensions of the grid (similar to MAZE WIDTH, see above)

(2) Output

After each avatar takes its turn, it appends data to the log file created in AMStartup (see above). Each line represents a move, and the log file catalogues successes and failures as they occur. See "Output" in the AMStartup design spec for more information about the log file name and contents. Avatar 0 also draws the maze, including traces, during its turn. This drawing is done using ASCII graphics and based on Avatar 0's knowledge of the maze.

(3) Data Flow

Each avatar is initialized by the AMStartup script using previously processed and stored data. When it is an avatar's turn (the avatar receives a valid AM_AVATAR_TURN message from the server), the avatar updates its grid using information from the AM_AVATAR_TURN message. It does this as follows (valid because AM_AVATAR_TURN contains the current position of each avatar):

1) If an avatar moved (current position is not equal to previous position), decrement the number of avatars in the old cell and increment the number of avatars in the new cell
2) If applicable, leave a trace in that avatar's old cell indicating which avatar left it and which way it was going
3) If there was already a trace in that avatar's old cell, replace that trace with the new trace

* See below for more information on traces - essentially, they serve as a heuristic to make cooperative maze solving easier. An avatar leaves a trace (a bread crumb or a piece of clothing) to indicate where it was and in which direction it went, and will follow a trace it encounters under certain circumstances.

After updating its grid (personal knowledge of the maze), the avatar determines its next move. The avatar first checks to see if there are other avatars on its space. If there are, it only attempts to move if it has the lowest ID. In determining its next move, the avatar follows several predetermined steps:

1) The avatar first looks for traces. If there is a trace on its current cell that is not its own (and that it does not recognize/know to ignore), it follows that trace by moving in the specified direction
2) If no trace is found, or the avatar knows not to follow the discovered trace, the avatar's move is determined using left-hand wall follow logic. If the avatar has just changed positions in the previous turn, it attempts to turn left

and move forward. Otherwise, it attempts to turn right and move forward (if it did not change positions, it either hit a wall or was not the lowest numbered avatar on its space - the latter case is irrelevant to maze solving)

3) The avatar makes its move, leaving a trace if applicable

Once the avatar has made a move, it sends information to the server (in the form of a AM_AVATAR_MOVE message containing a direction or M_NULL_MOVE) and its turn ends. This process continues until an exit condition (success or failure) is met, or until the number of expected moves is exceeded.

(4) Data Structures

The avatar program uses several data structures to keep track of its location and the locations of other traces/avatars:

Cell - stores information about an individual cell in the maze
- How many avatars are currently in that cell
- Direction of trace (NULL if no trace)
- Which avatar left trace (NULL if no trace)

Grid - multidimensional array of cell structs, represents maze
- Stored and updated by each avatar
- Cell* [width][height]
- [1][2] would represent row 1, column 2

XYPos - stores x and y coordinates of an avatar

PrevXY - array to store previous x and y positions of avatars
- XYPos* [numAvatars]
- Every index is avatar with that ID

ignoreList - list of trace IDs to ignore
- int [numAvatars]
- Always want to ignore your own traces
- If you're in the same cell as an avatar with a higher avatar ID, you want to ignore its traces
- This allows avatars on the same space to clump together and "follow" each other

prevMove – last valid move made by avatar
- Stored for use when turning right

(5) Avatar Pseudo-code

Initialize all structures and args
// Given from AMStartup

```
Once started, send AM_AVATAR_READY message containing avatarId
to server

While the maze is running, wait to receive a message:

     Message is AM_TOO_MANY_MOVES:

          Output failure and exit

     Message is AM_AVATAR_OUT_OF_TURN:

          Output failure and exit

     Message is AM_UNEXPECTED_MESSAGE_TYPE:

          Output failure and exit

     Message is AM_SERVER_TIMEOUT:

          Output failure and exit

     Message is AM_SERVER_DISK_QUOTA:

          Output failure and exit

     Message is AM_SERVER_OUT_OF_MEM:

          Output failure and exit

     Message is AM_AVATAR_TURN:

          If it is the current avatar's turn:
          // Determined by ID in message

               Update avatar's grid with new coordinates
               Place traces as necessary
               // Described above

               If other avatars on space:

                    Add avatars of higher ID to ignore list
                    // Allows avatars to follow each other

                    If not lowest ID:

                         Do not move
                         // M_NULL_MOVE
```

```
                If trace on current cell:

                     If not own trace or trace on ignore list:

                          Follow trace
                          // Move in trace direction

                No trace, so follow the wall:

                     If just changed positions:

                          Turn left and move forward

                     Else:

                          Turn right and move forward

                Leave a trace
                // Updated automatically

                Tell the server next move
                // AM_AVATAR_MOVE

                Write move to log file

           Done, move on to next turn
           // Resends AM_AVATAR_TURN

     Message is AM_MAZED_SOLVED:

           Output success message and exit
           // Only one avatar writes message

// Free memory
Clean up structures

Exit
```

* Note: For the purposes of unit testing, avatar functions have been placed into separate .c and .h files (avatarFunctions.c and avatarFunctions.h)