Team Oseleta
Amazing Project
Software Design Document

Janice Yip
Emily Greene
Lotanna Ezenwa
28 May 2014

Amazing Project Design Specification

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

1. *Input*
   Command Input for the Startup

   ./AMStartup [NAVATARS] [DIFFICULTY] [HOSTNAME]

   Example command input

   ./AMStartup 3 2 pierce.cs.dartmouth.edu

   [NAVATARS] -> 3
   Requirement: Must be positive number less than 10
   Usage: total number of Avatars generated for the maze

   [DIFFICULTY] -> 2
   Requirement: Must be an integer between 0 and 9 inclusive
   Usage: the difficulty level, on the scale 0 (easy) to 9 (excruciatingly difficult)

   [HOSTNAME] -> pierce.cs.dartmouth.edu
   Requirement: Must be a server
   Usage: server name we want IP address of

   Command Input for the client

   ./amazing_client [AVATARID] [NAVATARS] [DIFFICULTY] [IPADDRESS]
   [MAZEPORT] [FILENAME] [SHMID] [MAZEWIDTH] [MAZEHEIGHT]

   Example command input

   ./amazing_client 0 3 2 129.170.212.235 10829 Amazing_3_2.log 1234 20 20

   [AVATARID] -> 0
   Requirement: Number between 0 and max number of avatars
   Usage: an integer generated by AMStartup, starting at 0 and incremented
   by one for each subsequent Avatar started

   [NAVATARS] -> 3
   Requirement: Must be positive number
   Usage: total number of Avatars generated for the maze

   [DIFFICULTY] -> 2

Requirement: Must be an integer between 0 and 9 inclusive
Usage: the difficulty level, on the scale 0 (easy) to 9 (excruciatingly difficult)

[IPADDRESS] -> 129.170.212.235
Requirement: Must be a number
Usage: IP address of the server

[MAZEPORT] -> 10829
Requirement: Port number to connect to
Usage: MazePort returned in the AM_INIT_OK message

[FILENAME] -> Amazing_$USERID_3_2.log
Requirement: Writable file
Usage: Filename of the log the Avatar should open for writing in append mode.

[SHMID] -> 1234
Requirement: 4 digit number
Usage: Key for accessing shared memory

[MAZEWIDTH] -> 20
Requirement: Positive number
Usage: Maze's width, given by server

[MAZEHEIGHT] -> 20
Requirement: Positive number
Usage: Maze's height, given by server

2. *Console Output for Startup*
   The project returns an error if any error conditions are met (improper command line options, or the maze is not solved in time, wait time expires, or the connection to the server is lost), and success if the maze is solved (all of the avatars are determined to be at the same (x,y) coordinates).

   Graphics Output
   Maze printed using ASCII characters. Each MazeCell prints a wall, so each wall is double-thick. An empty maze is printed first, and then every print of the maze afterward includes the most recent location of each of the Avatars. Maze updates after each iteration where every Avatar is updated (i.e. every time that avatar 0 is updated).

   Log File Output
   The log file includes the user who ran the maze, the MazePort connected to the server, and the time that the test started as well as every avatar's move and the location to which the avatar moved. When the maze is solved, it prints the date and time along with the maze solved message.

Number of moves from some of our runs

Difficulty 1, 2 Avatars: 196
Difficulty 3, 3 Avatars: 4390
Difficulty 5, 4 Avatars: 11169
Difficulty 5, 5 Avatars: 7167
Difficulty 6, 5 Avatars: 13430
Difficulty 6, 6 Avatars: 22646
Difficulty 7, 4 Avatars: 19952
Difficulty 7, 5 Avatars: 25625

3. *Data Flow*

The AMStartup script connects to the server and sends a message to the server that specifies the desired number of Avatars and the difficulty of the maze. The server generates a new maze, sprinkles the Avatars across the maze, and responds with an AM_INIT_OK message. The AMStartup script takes the MazePort returned and starts N processes running the main client software.

Once started, each Avatar of amazing_client sends an AM_AVATAR_READY message containing its AvatarId to the server. When all the Avatars have sent the message, the server asks the Avatars for moves, and the Avatars send the moves.

This continues until the Avatar's socket connection is broken, the maximum number of moves is exceeded, the server's wait time expires, or all the Avatars have found each other.

Once the maze is solved, the server frees all the data structures relating to the maze and closes the MazePort. The Avatars then log their progress, close files, free allocated memory, and exit.

Data flow for graphics

Copies log.out off of the server using a system call.

Takes in the MazePort, MazeWidth, and MazeHeight in order to copy the log.out file from the server and create the visual representation of the maze.

Messages to and from the server from each individual avatar client

The client can send the server the following messages:

AM_INIT asks server to setup a new maze
AM_AVATAR_READY tells server that a new Avatar is ready to move
AM_AVATAR_MOVE tells server where an Avatar wishes to move

The server can send the avatar clients (each instance of amazing_client.c) the following messages:

AM_INIT_OK as a response stating initialization succeeded
AM_INIT_FAILED as a response stating initialization failed
AM_NO_SUCH_AVATAR as a response stating that the client referenced an unknown or invalid Avatar
AM_AVATAR_TURN as a response stating that the server updated Avatar (x,y) position and proceed to next turn
AM_MAZE_SOLVED as a response stating that the maze was solved
AM_UNKNOWN_MSG_TYPE as a response stating that there was a unrecognized message type
AM_UNEXPECTED_MSG_TYPE as a response stating that the message type was out of order
AM_AVATAR_OUT_OF_TURN as a response stating that the Avatar tried to move out of turn
AM_TOO_MANY_MOVES as a response that the run exceeded the max number of moves
AM_SERVER_TIMEOUT as a response that the run exceeded time between messages
AM_SERVER_DISK_QUOTA as a response that the server has exceeded disk quota
AM_SERVER_OUT_OF_MEM as a response that server failed to allocate memory

4. Data Structures

```
/* XY-coordinate position */
typedef struct XYPos
{
   uint32_t x;
   uint32_t y;
} XYPos;

/* Maze avatar */
typedef struct Avatar
{
   int fd;
   XYPos pos;
} Avatar;

/* AM Message description */
typedef struct AM_Message
{
   uint32_t type;

   /* Define a union for all the message types that have parameters. Messages
    * with no parameters don't need to be part of this union. Defined as an
    * anonymous union to facilitate easier access.
    */
   union
   {
      /* AM_INIT */
      struct
      {
```

```
    uint32_t nAvatars;
    uint32_t Difficulty;
} init;

/* AM_INIT_OK */
struct
{
    uint32_t MazePort;
    uint32_t MazeWidth;
    uint32_t MazeHeight;
} init_ok;

/* AM_INIT_FAILED */
struct
{
    uint32_t ErrNum;
} init_failed;

/* AM_AVATAR_READY */
struct
{
    uint32_t AvatarId;
} avatar_ready;

/* AM_AVATAR_TURN */
struct
{
    uint32_t TurnId;
    XYPos    Pos[AM_MAX_AVATAR];
} avatar_turn;

/* AM_AVATAR_MOVE */
struct
{
    uint32_t AvatarId;
    uint32_t Direction;
} avatar_move;

/* AM_MAZE_SOLVED */
struct
{
    uint32_t nAvatars;
    uint32_t Difficulty;
    uint32_t nMoves;
    uint32_t Hash;
} maze_solved;
```

```
        /* AM_UNKNOWN_MSG_TYPE */
        struct
        {
           uint32_t BadType;
        } unknown_msg_type;
    };
} AM_Message;
```

For graphics

```
/* Status for wall within MazeCell */
typedef enum {P, W} walls;

/* Each cell of the maze */
typedef struct MazeCell {
        XYPos position;
        walls north;
        walls south;
        walls east;
        walls west;
        int maze_boolean;
} MazeCell;

/* two-dimensional array of pointers to MazeCell structs */
MazeCell ***array;
```

For the algorithm

```
/* one-dimensional array of integers to determine presence of walls */
int *shared_mem;
```

5. Amazing Project Pseudo Code: Pseudo code description of the module

AMStartup.c

1. Check validity of arguments
2. Server is running, so run client start up script (AMStartup.c)
3. Send an AM_INIT message to the server at the AM_SERVER_PORT specifying the number of avatars and the difficulty of the maze
4. Receive a response from the server
    a. AM_INIT_OK: set up the MazePort specified

b. AM_INIT_FAILED: exit due to error

5. Set up shared memory of a 1-dimensional array of the maze. Each row of the maze is appended to the previous row in the array, such that the index of the cell is (yposition * mazewidth + xposition). Every index in the array is initialized to 0 to represent that there are no walls known.
6. N (number of avatars) processes are set up to run the main client software and each process has access to the shared maze.
7. Create a log document and parse the server's log document for graphics

****The user must enter their password in in order to access the log.out file

amazing_client.c

8. Each of the avatars will get a key to access shared memory
9. Each of the avatars will also get a char * filename of the log file they should append after they move
10. Each Avatar (instance of the client) sends an AM_AVATAR_READY message via the MazePort containing its assigned AvatarId to the server.
11. The server broadcasts an AM_AVATAR_TURN message to each avatar, telling the avatars whose turn it is
    a. Graphics are updated when one complete iteration of avatar moves have occurred, so every time the AM_AVATAR_TURN message is referring to avatar 0, all of the avatars will have been updated since the last call to graphics
    b. When graphics are updated, the function iterates through the AM_AVATAR_TURN message and plots the (x,y) coordinate of each avatar within the maze
12. On the first turn, each of the avatars calculate the central point of all the avatars, which is their end destination. Their last positions are set to (-1,-1), they are programmed to "face North," and they are assigned an array of direction priorities (based on the Right Hand Wall Following Algorithm).
13. After checking the avatar turn, the avatar whose turn it is gets the current position and compares it to the last one
    a. If the position is the same as the last one, increment the maze array for the avatar's current position to indicate there is a wall there. Also increment the maze

array of the other side of the wall. Each wall has a unique binary value (west is (2^0 = 1), north is (2^1 = 2), south is (2^2 = 4), and east is (2^3 =8)), so each combination of walls will also have a unique value. Then move on to the next direction the avatar should try to move in.

b. Otherwise, reset the direction priority array based on which direction the avatar is currently facing.

14. The avatar requests its next move from the server using a AM_AVATAR_MOVE message. If he has already found the calculated centroid, don't move
15. If the move is allowed, the server updates the avatar's position. If the move is not allowed, the avatar stays in the same position
16. The server sends out an updated AM_AVATAR_TURN message
17. This process will continue until one of the following occurs:
    a. an Avatar's socket connection to the server is broken,
    b. the maximum number of moves (a function of AM_MAX_MOVES and Difficulty) is exceeded,
    c. the server's AM_WAIT_TIME timer expires, or
    d. the server determines that all of the Avatars are located at the same (x,y) position, meaning the maze has been solved.
18. When the maze is solved, the server sends an AM_MAZE_SOLVED message to all of the Avatars. The server then frees all the data structures relating to this maze and closes the MazePort. Upon receiving this AM_MAZE_SOLVED message, the Avatars ensure that the AM_MAZE_SOLVED message is written to a log file once.
19. Whether the maze is solved or one of the other exit conditions occurred, the Avatars should log their success/progress, close any files, free any allocated memory, etc., and then exit

Amazing Project Implementation Specification

1. Major data structures and important global variables

```
// define how long our IP Address can be
        #define MAX_IP_LEN 100
```

```
// define how long the filename can be
        #define MAX_FILE_NAME 100

// define how long the id can be
        #define MAX_ID_LEN 5

// define how long port length can be
        #define MAX_PORT_LEN 10

// define how long key length can be
        #define MAX_KEY_LEN 10

// define max maze width
        #define MAX_WIDTH 102

// define max maze height
        #define MAX_HEIGHT 102

// define 2^0 for west wall
        #define W_WALL 1

// define 2^1 for north wall
        #define N_WALL 2

// define 2^2 for south wall
        #define S_WALL 4

// define 2^3 for east wall
        #define E_WALL 8
```

2. Prototype Definitions

```
/*
 *  File functions within AMStartup.c
 */

/*  IsNotNumeric - Checks input is a number, return 0 on success; -1 otherwise  */
int IsNotNumeric(char *filename);

/*
 *  File functions within amazing_client.c
 */

/*  IsNotNumeric - Checks input is a number, return 0 on success; -1 otherwise  */
int IsNotNumeric(char *filename);
```

```
/* Has North Wall - Checks whether the current maze cell has a north wall */
int HasNorthWall(int index);

/* Has West Wall - Checks whether the current maze cell has a west wall */
int HasWestWall(int index);

/* Has South Wall - Checks whether the current maze cell has a south wall */
int HasSouthWall(int index);

/* Has East Wall - Checks whether the current maze cell has a east wall */
int HasEastWall(int index);
```

Summary of Error Conditions Detected and Reported

Errors that kill AMStartup.c program:

1. If the number of avatars is not a number, program prints "Number of avatars must be a number. Exiting now." and exits.
2. If the difficulty is not a number, program prints "Difficulty must be a number. Exiting now." and exits.
3. If the socket cannot be created, program prints "Problem creating the socket." and exits.
4. If the socket cannot connect, program prints "Problem connecting to the server." and exits.
5. If the socket is not received properly, program prints "The server terminated prematurely." and exits.
6. If the shared memory is created incorrectly, program prints "shmget failed", and exits.
7. If the return message is not what is expected, program exits.

Testing

1. Simple argument checks are done upon initialization of each avatar (in amazing_client.c) as well as in the main AMStartup.c file
2. Unit tests were performed on the functions used in maze.c
    a. For the parselog function, we tested what would happen if the file was not found
    b. Tested to make sure the log file was read and the maze was created
    c. Tested the freeing of the memory used by the array
3. Valgrind was used to find memory leaks
4. Upon using valgrind, we refactored the code to more efficiently use heap memory
5. Afterwards, we were able to implement freeing of the memory used by the array