# Lambrusco Amazing Project Design Document

COSC 50 Professor Palmer 16W

Kyu Kim, Alan Lu, Donald Wilson

## AMStartup Program Design Specification

**Goals:**

The startup program should communicate with the server and take parameters necessary to launch the appropriate number of Avatars in a maze generated by the server with the difficulty (i.e. have nAvatars and Difficulty as arguments to script). The program should create a log file to document what each Avatar is doing and the general activities of the maze. The script will then start up N Avatar processes and let them run until the socket connection is broken, max # moves is exceeded, the timer expires, or the maze is solved.

**Inputs:**

Command Input:

./AMStartup [nAvatars] [Difficulty] [hostName]

Example:

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

[nAvatars] (uint32_t)- number of Avatars to be placed inside maze
Requirement: Must be int between 2 and AM_MAX_AVATAR (which is 10)
Usage: AMStartup must inform user if nAvatars is not between 2 and 10.

[Difficulty] (uint32_t)- difficulty of the maze
Requirement: Must be int between 0 and 9
Usage: AMStartup must inform user if Difficulty is not between 0 and 9.

[hostName] (char *)- name of the host
Requirement: Must be flume.cs.dartmouth.edu
Usage: AMStartup must inform user if hostName is not flume.cs.dartmouth.edu

**Outputs:**

The script will startup N Avatars as processes and place them inside the maze generated by the server at the MazePort. The server will return to AMStartup in a AM_INIT_OK message: a unique MazePort, MazeWidth, and MazeHeight. The MazePort will be used as an argument for generating and placing the Avatars.

The script also creates a log file in the format Amazing_$USER_N_D.log, where $USER is the current userid, N is the value of nAvatars and D is the value of Difficulty. It then writes the first line of the log which should contain $USER, the MazePort, and the date & time using time() and ctime() functions in time.h.

**Data Flow:**

The variables nAvatars and Difficulty are both stored in variables of type uint32_t to be used for starting up the Avatar clients. These variables will be used in the AM_INIT message sent to the server at AM_SERVER_PORT (a constant). The server will return an AM_INIT_OK message if it is ready with the MazePort, MazeWidth, and MazeHeight.

MazePort, MazeWidth and MazeHeight will also be stored in variables of type uint32_t. MazePort will be used in initializing the Avatar clients. The new log file that will be created will use the variables $USER (from shell), nAvatars and Difficulty as part of the name.

**Data Structures:**

All data that needs to be stored in AMStartup does not require a major data structure that is complex in nature.

nAvatars (uint32_t)- number of Avatars
Difficulty (uint32_t)- difficulty of maze
MazePort (uint32_t)- Port number that server returns
MazeWidth (uint32_t)- Maze width that server returns
MazeHeight (uint32_t)- Maze height that server returns

**Pseudo-code:**

// check command line arguments and inform user if invalid arguments

// send AM_INIT message to server at AM_SERVER_PORT specifying nAvatars (number of avatars in maze) and Difficulty (difficulty of maze)

// check AM_INIT_OK message, get MazePort, MazeWidth, and MazeHeight from message

// creates a new log file with the name Amazing_$USER_N_D.log, where $USER is the current userid, N is the value of nAvatars and D is the value of Difficulty.

// write first line of log which includes: $USER, Mazeport, date & time

// startup N processes (one for each Avatar) with appropriate startup parameters (AvatarID (starting at 0, incrementing by 1), nAvatars (total # Avatars), Difficulty, IP address of server, MazePort, FileName of log avatar should open in append mode

// let client run until Avatar's socket connection is broken, max # moves (a function of AM_MAX_MOVES and Difficulty) exceeded, server's AM_WAIT_TIMER expires, or maze is solved

# Avatar Program Design Specification

**Goals:**

The Avatar client will take inputs from AMStartup and the server and use them to generate a process that can communicate with the server. Each Avatar will communicate with the server to move in a turn-based manner. The way that each Avatar moves will be determined by the maze-solving algorithm. This process continues until the Avatars are all in the same location.

**Inputs:**

Command Input

./amazing_client [AvatarId] [nAvatars] [Difficulty] [IP] [MazePort] [Filename]

Example

./amazing_client 0 3 2 129.170.212.235 10829 Amazing_3_2.log

AMStartup will provide the following parameters when initializing an Avatar process:

1. AvatarId (an integer between 0 and (nAvatars - 1), inclusive)
2. nAvatars (total number of Avatars between 2 and AM_MAX_AVATAR)
3. Difficulty (difficulty of the maze between 0 and 9)
4. IP address of the server
5. MazePort (a TCP/IP port returned to AMStartup by the server)
6. Filename of the log the Avatar should open for writing in *append mode*

The Avatar can expect to receive the following message types from the server:

| Message | Summary |
|---|---|
| AM_INIT_OK | response: initialization succeeded<br>Returns MazePort, MazeWidth, and MazeHeight |

| | |
|---|---|
| AM_INIT_FAILED | response: initialization failed<br>Returns ErrNum |
| AM_NO_SUCH_AVATAR | response: referenced an unknown or invalid Avatar<br>Returns AvatarId |
| AM_AVATAR_TURN | response: updated Avatar (x,y) position and proceed to next turn<br>Returns TurnID and Avatar positions |
| AM_MAZE_SOLVED | response: the maze was solved<br>Returns nAvatars, Difficulty, nMoves, and Hash |
| AM_UNKNOWN_MSG_TYPE | response: unrecognized message type<br>Returns BadType |
| AM_UNEXPECTED_MSG_TYPE | response: message type out of order |
| AM_AVATAR_OUT_OF_TURN | response: Avatar tried to move out of turn |
| AM_TOO_MANY_MOVES | response: exceeded the max number of moves |
| AM_SERVER_TIMEOUT | response: exceeded time between messages |
| AM_SERVER_DISK_QUOTA | response: server has exceeded disk quota |
| AM_SERVER_OUT_OF_MEM | response: server failed to allocate memory |

(from assignment description)

**Outputs:**

The Avatar will communicate the following message types to the server:

| Message Type | Summary |
|---|---|
| AM_INIT | asks server to setup a new maze<br>Parameters: nAvatars and Difficulty |
| AM_AVATAR_READY | tells server that a new Avatar is ready to move<br>Parameters: AvatarId |
| AM_AVATAR_MOVE | tells server where an Avatar wishes to move<br>Parameters: AvatarId and Direction |

(from assignment description)

**Data Flow:**

Upon receipt of the different input variables ( [AvatarId] [nAvatars] [Difficulty] [IP] [MazePort] [Filename] ) from AMStartup, it will store these in simple variables for use in messages sent to the server. Generally, the Avatar will communicate solely with the server until the maze is completed.

Each Avatar will send a AM_AVATAR_READY message to the server. When the server receives all the ready messages, it will send an AM_AVATAR_TURN message to every Avatar indicating each Avatar's position in the maze and its TurnID. Using this information, the Avatar will return to the server an AM_AVATAR_MOVE message with the move it wants to make using our maze-solving algorithm. The server will determine if the move is valid and return an AM_AVATAR_TURN message to all the Avatars. If the (x, y) coordinate of the Avatar is updated, the move was legal and the next Avatar can go through the same process of trying to make a move.

The communication between the server and Avatars will also make use of the network byte ordering functions:

```
#include <netinet/in.h>

// Host uint32 to network uint32
uint32_t htonl(uint32_t host32bitvalue);
// Network uint32 to host uint32
uint32_t ntohl(uint32_t net32bitvalue);
```

**Data Structures:**

Most of the information for Avatars is stored in relatively simple variables that do not require explanation.

2-Dimensional Linked List (A list of lists) - For keeping track of the walls of the graph for graphics and possibly maze-solving algorithm later.

A 2-D Linked List seems simple enough to easily implement and will be efficient for printing out for graphics.

**Pseudo-code:**

```
// Check args (checking number of args is sufficient)

// Initialize local variables
// Set i = 0                    // count of unsuccessful moves
// Set dir = 1                  // direction of last successful move
// Set nextDirection = 0        // direction to attempt next

// Initialize an Avatar struct, setting fd = AvatarID, pos = NULL

// Send AM_AVATAR_READY message

// While we haven't completed the maze:
    // Listen for a message from the server

    // If AM_AVATAR_TURN:
        // If it's my turn to move (i.e. TurnID == AvatarID):
            // If my position changed:
                // Update position
                // Update direction of last successful move:
                // Set dir = nextDirection
                // Set i = 0 (0 indicates a left turn; 1 ahead; 2 right; 3 back)

            // (Note: We designate Avatar 0 as the "exit")
            // If position == exit location (Avatar 0's position):
                // Stay put; set nextDirection = 8 (null move)
                // Call Move function: Move(nextDirection)
            // Else
                // Set nextDirection = (direction + 3 + i) mod 4
```

```
                    // Attempt to make a move in the nextDirection
                    // Increment i: i++

        // Else if AM_MAZE_SOLVED
                // Avatar 0 writes a success message to log file
                // Close files, free allocated memory, etc
                // Exit

        // Else if AM_AVATAR_TOO_MANY_MOVES or AM_SERVER_TIMEOUT or
        // AM_SERVER_OUT_OF_MEM
                // Avatar 0 writes message to log file
                // Close files, free allocated memory, etc
                // Exit

        // Else if AM_NO_SUCH_AVATAR or AM_UKNOWN_MSG_TYPE or
        // AM_UNEXPECTED_MSG_TYPE or AM_AVATAR_OUT_OF_TURN
                // Write error message to log file

        // Else if AM_SERVER_DISK_QUOTA
                // Write error message to log file
                // (Note: Please notify TA/instructor in this event)

// Exit
```