EE 306: Introduction to Computing

Programming Lab 4

Course Instructor: Prof. Nina Telang
TAs: Aniket Deshmukh, Shruthi Krish

All Lab assignments must be completed individually. You are not permitted to seek help or clarification from anyone other than the instructor or the TAs.

Due date: 4/22, 11:59 PM

Note that this lab needs to be checked out by the TAs. This will be done during the office hours on the following days (4/23, 4/24).

Problem Statement

For this lab, you will be writing code for a game most of you might be familiar with, Pac-Man! This is a toned-down version of the full game, with an option of scaling it up for extra credit.

The base version of the game contains a maze created using a string of characters ('*****') that is displayed on the console. Pac-Man (character '**<**') starts at the top left corner and waits for user input. Using the '**w**', '**a**', '**s**' and '**d**' keys, the user directs Pac-Man to move a single step in the corresponding direction. The goal of the game is to eat the dot (only one, character '**o**') on the screen, while avoiding the ghosts (character '**U**'). The game ends when the positions of the Pac-Man and either the dot, or one of the ghosts are the same. Eating the dot prints the message, '**Game Over, You Win**!'. Eating a ghost however results in the message, '**Game Over, You Lose**!' being printed. In both cases, the processor halts after the message is printed.

For this lab, you will be given some starter code that contains a few subroutines that may be useful. The document will also go through the general flow of the program, including an exact description of the subroutines you need to provide to complete it. We strongly recommend that you follow the flow described here. However, if you wish to write the program on your own (from scratch or modify the subroutines) you are welcome to do so.

Gameplay Rules and the Maze

As mentioned earlier, Pac-Man navigates through the maze based on user inputs. '**w'** is UP, '**a**' is LEFT, '**s**' is DOWN and '**d**' is RIGHT. To emulate animation of ghosts, the maze is represented by a set of frames stored in memory. Each frame consists of a 46x23 grid, where a row of length 46 is stored using a .STRINGZ (23 characters and a null). To get a square shaped maze we add a space between the characters in a row (which is why it has 45 characters). The maze frames are stored beginning at location x5000. Therefore, the first row of the first frame is at address x5000. Observe the file, 'maze.asm' that has been provided. At x5000, there is string of alternate '*' and ' ' characters stored (ASCII x2A, x20). There are 23 such rows in one frame. The '*****' characters form the boundaries and the walls of the maze

which **should not** be modified, i.e., moving into the wall results in no change in the position of the Pac-Man

To see an example of how one would index into the maze, look at the INIT and DISPLAY subroutines which have been provided to you.

Every time the user enters an input, the pointer to the maze and the position of Pac-Man is updated to point to the next frame. Each consecutive frame contains the ghost moved by a single character, and thus the ghosts move along with Pac-Man.

Subroutines

(Fig. 1)

INIT

- Initializes the maze pointer with the first frame and Pac-Man's initial position

This subroutine has been provided to you. The maze pointer is the address of the first frame. In the sample code provided, a memory location is used to store this pointer. Pac-Man's initial position is the top left corner. Since the edges have the border (character '*****') the initial head position is the 2nd row, 2nd column. To get to the next column, we increment the address by #2. To get to the next row, we add #46 (length of a single row).

DISPLAY

- Displays all the character in the current frame.

This subroutine has been provided to you. From the base pointer, all 23 rows are displayed on the screen. You can use this as an example to see how each frame is indexed.

GET INPUT

- Get user and input and save it.

You need to write this subroutine. All this subroutine does is wait for the user input (one character) and saves it (either to a memory location or register).

UPDATE HEAD POSITION

- Based on the user input, calculate the new position of the head.

You need to write this subroutine. This subroutine uses the input from the previous subroutine. If the user entered command was one of 'w', 'a', 's' or 'd', the head position is updated to one character ABOVE, LEFT, BELOW or the RIGHT of the current position. If any other button is pressed, the position is not updated. The position is also not updated if the character at the new position happens to be a wall (the character '*****'). Thus, Pac-Man does not move if a command entered causes him to run into a wall!

UPDATE MAZE

- Update the maze pointer to point to next frame, and the head position to point to the same location in the next frame.

This subroutine has been provided to you. Since the user input has been taken and the head position changed, the positions of all the ghosts on the maze need to be changed too. This is done by modifying the maze pointer to point to the next frame. Also, the head pointer needs to be updated to point to the position of the head on the next frame. This subroutine cycles through the frames starting from the first frame to frame number 23. After reaching frame 23, the first frame is loaded again.

MOVE HEAD

- Based on the old and new head position pointers, move the character corresponding to the head to the new position.

You need to write this subroutine. Now that a new frame is loaded, the actual character corresponding to the head is moved from the earlier frame to the new one. The character that is overwritten in the new frame is also recorded. Also, modify the Pac-Man character on the earlier frame appropriately (either a space ‘ ‘ or a ghost ‘**U**’);

CHECK END

- Check the character at the updated head position, and halt if match.

You need to write this subroutine. After moving the head if the previous character was an ‘**o**’, then the message ‘Game Over, You Win!’ is printed. If the character was a ‘**U**’, then the message ‘Game Over, You Lose!’ is printed. Otherwise, loop back to getting the input.

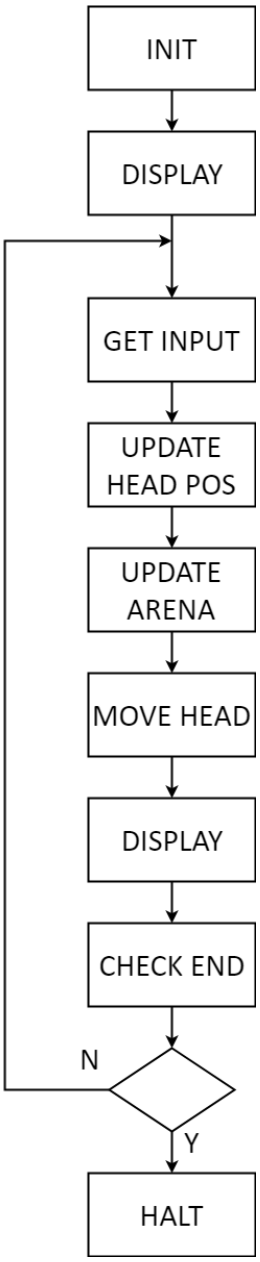
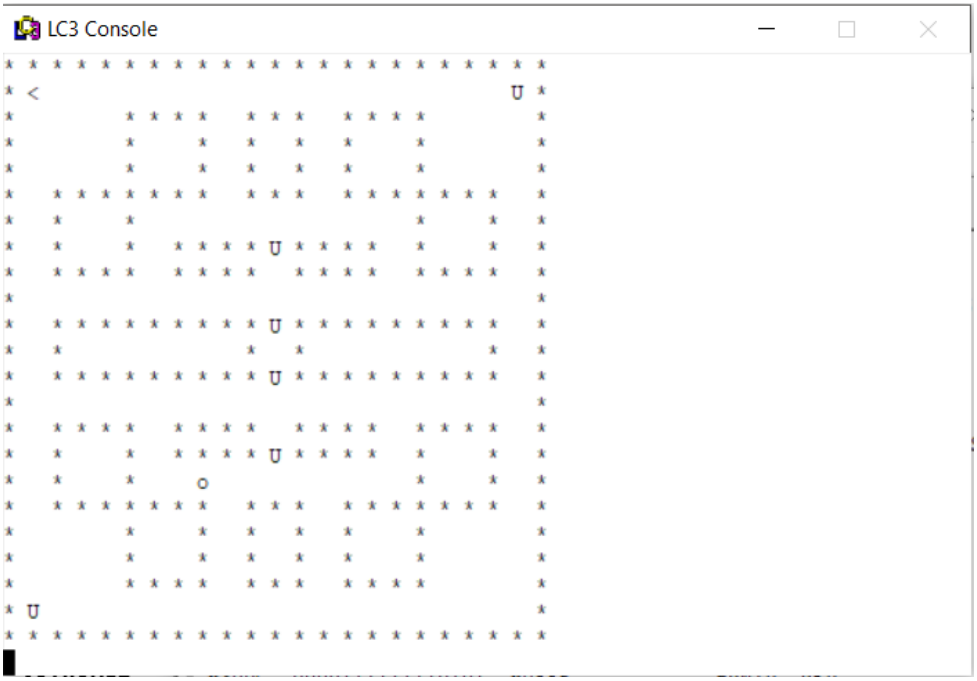


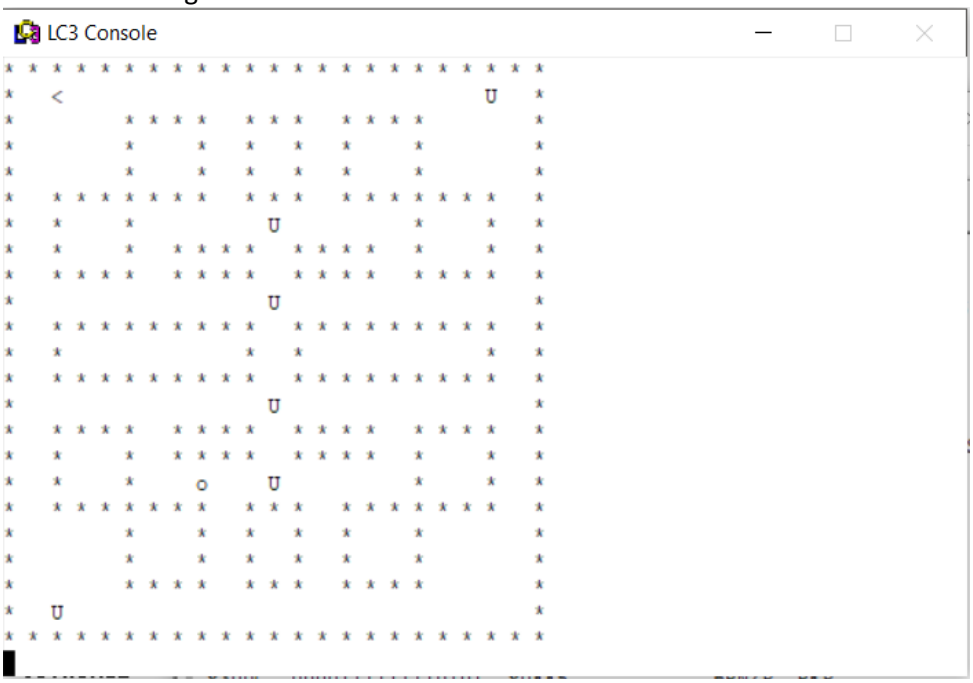
Fig. 1: Program Flow

Example run of the program

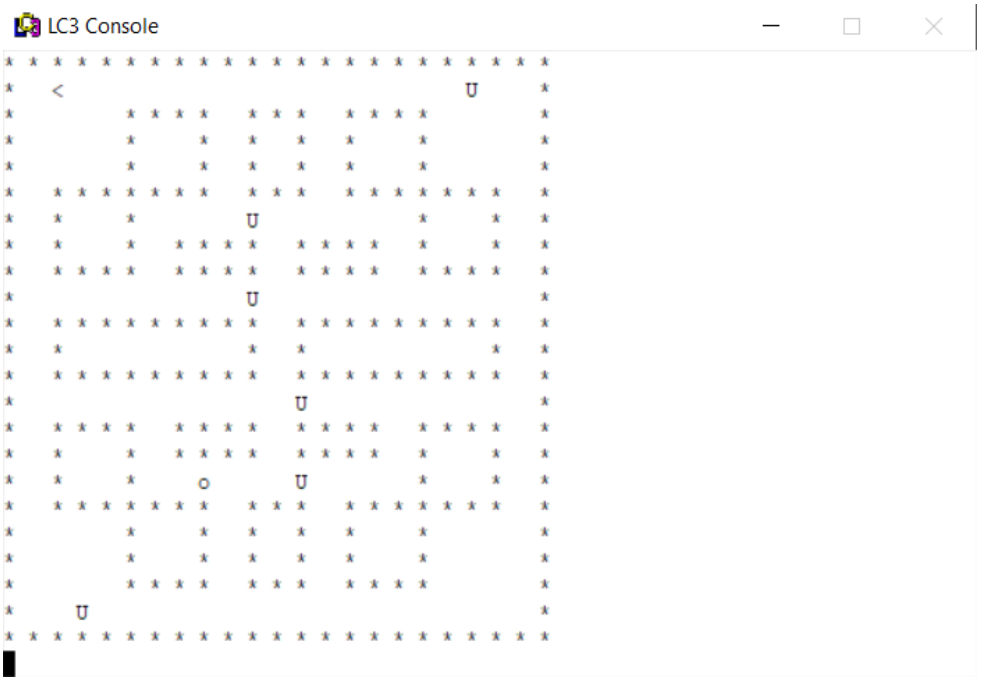
1. Initial State of the Maze



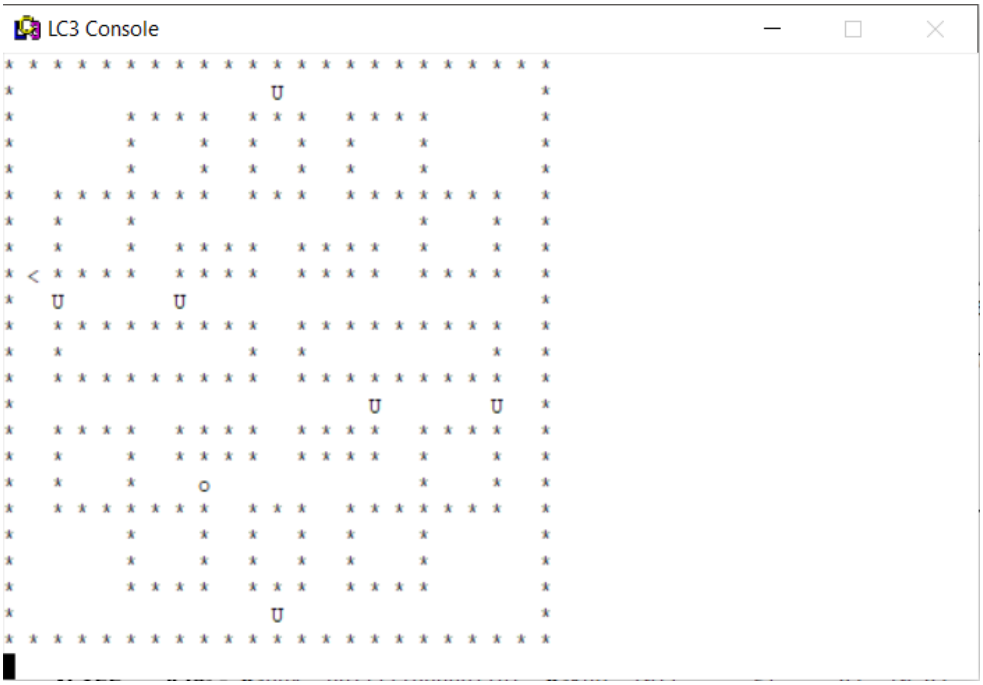
2. Pressing '**d**' once



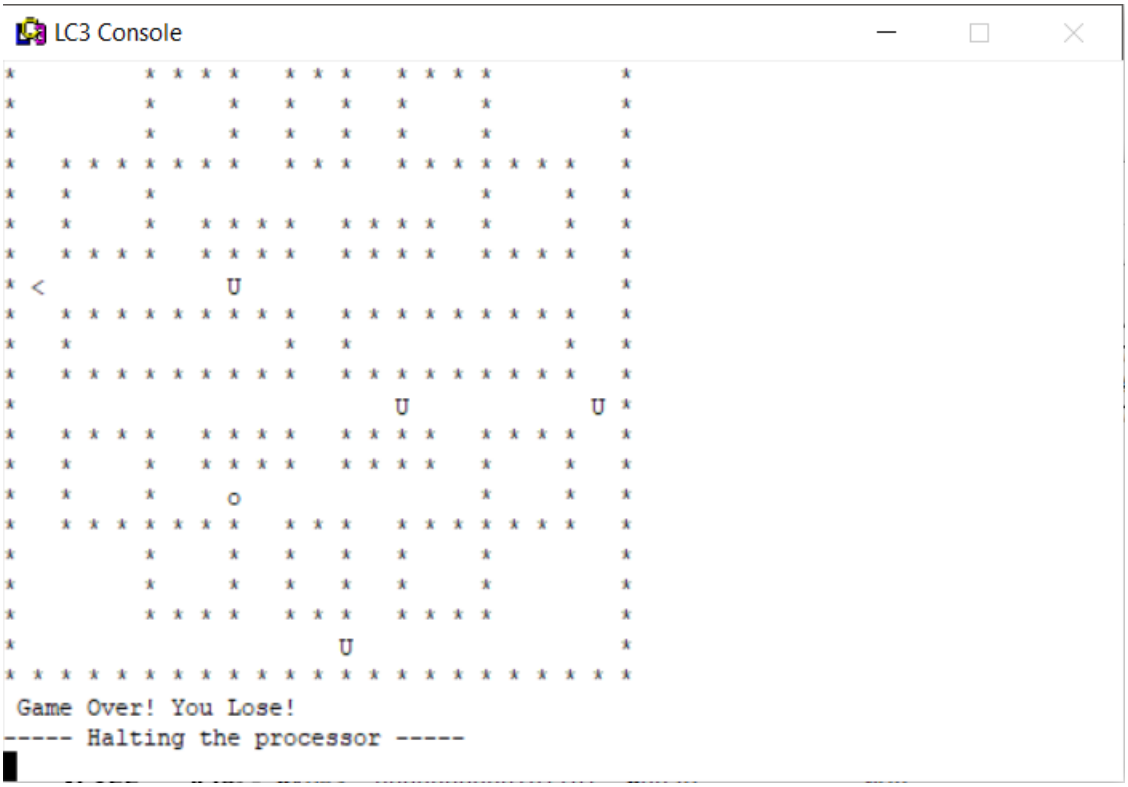
3. Pressing '**w**' once. Pac-man does not move, but the arena is updated, and the ghosts move.



4. After a sequence of moves, Pac-Man is near a ghost



5. The user enters '**s**'. Both Pac-Man and the ghost move into each other, and the game ends.



EXTRA CREDIT

The game is won when only a single dot is eaten. In the original game, all dots on the screen need to be eaten to win the game. For extra credit, you are required to modify the arena to include multiple dots. You code should now halt only when all dots are consumed. Along with the original message, you also need to print the score, i.e., the number of dots that have been consumed. Note that once a dot is eaten, you need to modify **all** the frames to remove the dot from the maze.

Notes

- As you must have realized, this lab is quite complex. In addition to the code, a major portion of this lab involves **reading** the document and the code provided to you to complete the program.
- Pay attention to the inputs and output of your subroutines. You are likely to run out of registers and using memory locations for certain variables is advisable.
- You can change the maze by modifying the file 'maze.asm' if you wish to do so. However, note that the provided subroutine will then also have to be modified.
- Pay attention to the row and column offsets, and the fact that there is a space between characters in the same row.
- You are free to make changes in the positions of the ghosts, the dot and the walls as long as the game mechanics (don't move into a wall, eating a ghost ends the game) are adhered to.