Assignment 2 Data Structures 2023
Task Details
Quick Preview
First of all, please watch the supplementary videos on the Assignment link on Blackboard. These videos demonstrates what we expect your program should do. You will expand the basic functionalities of the maze program you did for assignment 1 (The ”limited visibility” feature is not needed to complete assignment 2). Further details on the specification will be explained on the oncoming sections.

Command Line Arguments
Please ensure that the executable is called “maze”. Your executable should accept one commandline
parameters/arguments:
./maze <map_file_name>
Or
dir> maze <map_file_name>
<map_file_name> is the textfile name that contains the metadata about the map. You have to use the proper File I/O functionalities to retrieve the information. For more detail, please refer next section.
If the amount of the arguments are too many or too few, your program should print a message on the terminal and end the program accordingly (do NOT use exit() function). If the file does not exist, then you need to handle it with a proper error message.

File I/O and metadata
Please watch the supplementary video about the File I/O. Do you remember when you need to retrieve the metadata from the getMetadata() function in assignment 1? Now, you will retrieve the metadata from the textfile. This is one example:

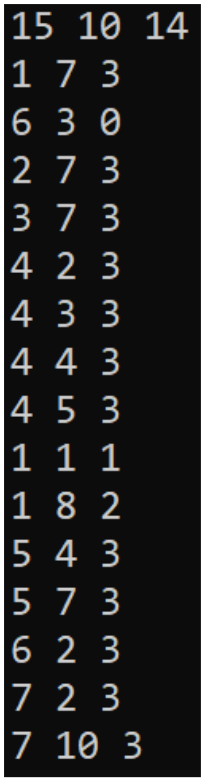
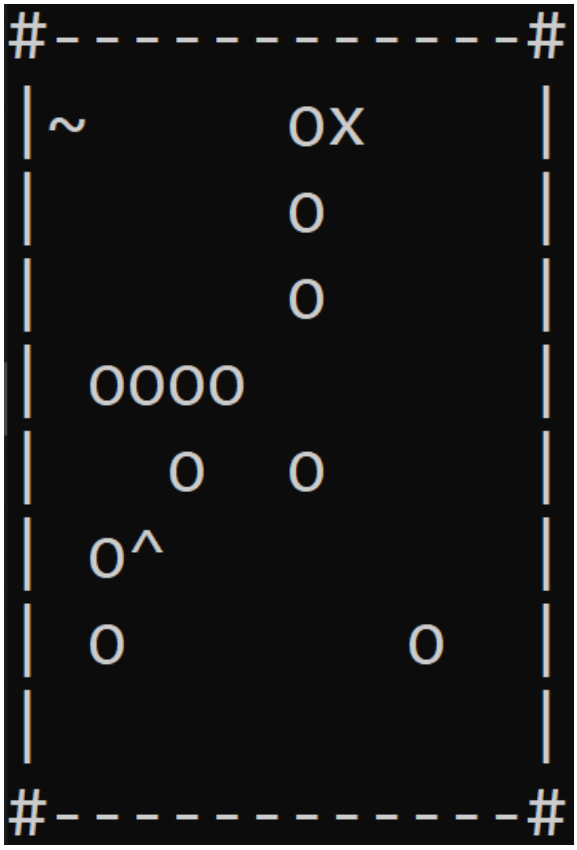


Figure 1: Example of a text file containing the map metadata.

There are always 3 integers on every line. The first line always contains the <metadata_amount>, <map_row>, and <map_col> in that order. From the example of figure 1, the first line means that there are 15 objects on the map of size 10 rows and 14 columns. The following lines are the coordinates of the object with the pattern <object_row>, <object_col>, and <object_code>.
The <object_row> and <object_col> are the zero-based index location of the objects (For example, <object_row> of 2 refers to the third row of the map). Please pay special attention on the <object_code> since it is slightly different from assignment 1:
• 0 – is the player that you control.
• 1 – is the snake that will try to bite the player.
• 2 – is the goal you want to reach to win the game.
• 3 – is the wall. Player and snake cannot go through it.
Unlike assignment 1, there is NO guarantee that the player and the goal would be the first two objects to be listed. It means that the coordinate of the player, snake, and the goal can be on any line. Therefore, your implementation should be able to handle that. However, you can assume that the amount of the objects matches the <metadata_amount> value, and there is exactly one player, one snake, and one goal.
Note: Since the edge of the map will contain the impassable border, the playable area size of the map is (<map_row>-2) x (<map_col>-2).
Note: Keep in mind that we will use our own map text file when we mark your assignment. So, please make sure your file I/O implementation works.

Main Interface

Similar to assignment one, your program should clear the terminal screen and print the 2D map: Then the user can enter the command via key ’w’, ’a’, ’s’, and ’d’ to control the player. Every time



the user inputs the command, the program should update the map accordingly and refresh the screen (clear the screen and reprint the map).

Undo feature with Generic Linked List

In addition to the command to move the player, you will add another command with key ’u’that allows the player to undo the steps taken when the player moves. Your program should utilize linked list to keep track all the steps taken since the beginning of the game. If the player attempts to walk to the wall, that is also recorded in the linked list. Please watch the supplementary video on this topic.

Please make sure you have a decent understanding on linked list practical exercises before you implement this feature. Implementing insertFirst() and removeFirst() function will be useful for the undo feature. In order to get full mark on this feature, you need to implement the generic linked list. If you implement the non-generic linked list, you still can receive decent partial marks.

Snake Movement and Losing condition

In this assignment, we introduce a new character ’_’ (tilde character) to represent the snake on the map. This snake will gradually go closer to bite the player. Once the player and the snake is on the same location, the player loses the game. Please watch the supplementary video on this topic.

The snake will attempt to move one step closer every time the user enters one movement command key (not including undo key). The snake cannot move in the diagonal direction. If the player goes towards the wall, that also counts as a single movement and the snake will move one step as well. The snake also cannot go through the wall. You are free to design the algorithm that makes the snake move. The snake does not have to be very ”smart” (can get stuck on the wall), but it should be reasonable so that it attempts to get closer to the player every time.
Note: When the snake bites the player, the player cannot use the undo feature anymore.

Assumptions

For simplification purpose, these are assumptions you can use for this assignments:
• The metadata amount will always match the amount of the objects.
• The coordinates of all objects provided in the text file are all valid. (NOT out of bound)
• There will be exactly one player, one snake, and one goal from the metadata. (but the order in the text file is not guaranteed)
• The size of the map will be reasonable to be displayed on terminal. For example, we will not test the map with gigantic size such as 300 x 500. It is the the same for the opposite scenario, we will not test your program with the map size less than 5 x 5.
• When we mark your assignment with our own map text file, you can assume the path from the player to the goal exist. So, it means the game is winnable. Similarly, you can assume the snake will have a path to reach the player. (not isolated)
• You only need to handle lowercase inputs (’w’, ’s’, ’a’, ’d’, ’u’).

• **FUNCTIONALITIES**:
– Correct command line arguments verification with proper response.
– Correctly handles the file I/O to open and read the provided text file to retrieve the metadata. This includes being able to read the location of the player, snake, and the goal correctly despite being located on random lines.
– Implement the generic linked list and store both the player and snake’s location every time the player receives the command to move.
– Able to ”undo” the player and snake location based on the information stored in the linked list. By pressing ’u’ multiple times, it is possible to go back up to the original map state.
– The snake is able to move towards the player on every player movement.
– Losing the game when the snake bites the player OR the player steps on the snake. (both objects are on the same coordinate)