Set up a new project in Eclipse and add these files to it.

mazegame.py is a program that implements the beginnings of a game about finding a way

through a maze. When you run this program, you will see a screen, 1000 pixels wide by 500

pixels high, displaying a maze. In our game, the maze is a two-dimensional structure with

walls and paths. A character, called Theseus, is placed somewhere in the maze and has to

follow the paths to achieve the goals of his missions. The paths can branch, which means that

Theseus has choices where to go next. As for his missions, he has to achieve the following:

1. Walk through the maze in a systematic way such that eventually he has visited all

locations that are reachable from his initial position.

2. Find his way out of the maze.

3. Find the quickest way out of the maze.

To achieve the first mission, Theseus performs a depth-first search from his current location,

and after he has done so, returns to his current position. He marks the locations that he has

visited with a special token, and the ones he has revisited with another special token.

For the second mission, Theseus traverses the maze in his mind without actually marking

locations – he has a good memory of the maze – to find his way to the exit. When doing so,

he has two choices. He can record the path taken on a stack, which he then uses to build the

path to actually walk to the exit, or alternatively, he can set a parent attribute for each cell in

the maze that he visits, then once he finds the exit, he can follow the parent references to

build the path that he uses to walk to the exit.

For the third mission, Theseus traverses the maze in his mind again. To find the shortest path

to the exit he can use a breadth-first search, where he records the cells that he visits in a

queue. Each cell added to the queue should have the parent attribute set to be location of thecell that caused it to be added to the queue. Once Theseus finds the exit, he can follow the

parent references to build the shortest path to the exit. Alternatively, Theseus can search

recursively and set a cost attribute for each cell that he visits. This records the shortest

distance that he has had to travel to reach the cell. Whenever he comes upon a cell that he

has visited previously, he compares the cost value for the cell with his current path cost. If

his current path cost to the cell is smaller than that recorded, he updates both the parent

reference and the cost for the cell and continues his search from that cell. Otherwise, if his

current path cost to the cell is larger than or equal to that recorded, he ignores the cell as he

continues his recursive search. Once Theseus has completed his search, he can again follow

parent references to build the shortest path to the exit.

mazeclass.py uses a two-dimensional grid (list of lists) to represent the maze. Each list item

is a cell, with attributes status, parent and cost. The values for the status attribute have

the following meaning:

0

The location is part of a path and has not been visited by Theseus before.

1

The location is part of a wall and therefore cannot be visited.

2

The location is part of the perimeter of the maze and therefore cannot be visited.

3

The location is part of a path and has been visited by Theseus.

4

The location is part of a path and has been revisited by Theseus (i.e., visited for a

second time).

5

The location is part of a path and has been examined by Theseus in his mind.

6

The location is the exit square.

The constructor for a maze uses a two dimensional integer array of status values to build the

maze, and places Theseus at a random location in the maze. To test your program using

different mazes you can insert different two dimensional integer arrays as values for the

mazegrid variable in mazegame.py. The code has a number of methods that are used to

display the maze, which you can use as is, or alter to your liking.

mazegame.py includes headers for the three functions that you are supposed to implement:

def4dfs(i,4j):

444...

This function performs the depth-first search and changes the entries in the maze grid

appropriately. It also takes care of displaying the maze and providing a time delay after each

change. An easy way to do this is by including the following code:the\_maze.display\_maze(screen)

pygame.time.delay(200)

pygame.display.flip()

def4setpath(i,4j):

444...

This function looks for the exit in the maze without initially changing the display of the

maze. It changes the entries for the locations in the maze that have been examined from 0 to

5. It also keeps track of the path, which is then displayed step by step at the end of the

procedure.

def4shortestpath(i,4j):

444...

This function looks for the shortest path to the exit in the maze without initially changing the

display of the maze. It also keeps track of the path, which is then displayed step by step at

the end of the procedure.

You can invoke these functions by pressing the 'd, 's' or 'q' keys, respectively. The main game

loop takes care of recognizing key presses and allowing to quit the program. The maze is

reset to its initial state before each invocation of one of the three functions.