The files included are as below:

1. searchAgents.py

2. search.py

3. dfs.P

4. bfs.P

5. Astar.P

6. CornersProblem.P

7. maze.P

As mentioned in the problem the maze.P file is created by giving each of the corordinates a unique number and the file is dumped with the connections where there is no obstruction. The format of the connection is connect(x,y,z) where x,y represents the unique coordinate in the maze which has no barrier i.e wall in between them. The thrid argument z represents the direction the pacman has to move inorder to reach the next coordinate.

1. searchAgents.py:

This file includes the code that is modified under class : PositionSearchProblem. The code deals with writing all the possible connections that can happen in the maze into the maze.P file. The connections are represented in the format : connect(x,y,z) where x,y represents the unique coordinate in the maze which has no barrier i.e wall in between them. The thrid argument z represents the direction the pacman has to move inorder to reach the next coordinate. The goal(v) and start(u) represents the goal of the pacman where the good is avaliable and the start of the pacman respectively. This file maze.P will be given as input to the prolog file to analyze the moves and come out with the solution that gives the list of the path to move to the goal coordinates. The analysis can be done using breadthFirstsearch or depthFirstSearch or A\* algorithm.

2. search.py

This file includes the code to call the prolog files in which the various algorithms have been implemented. The files that are being called are dfs.P, bfs.P and Astar.P. The code to integrate the call to prolog file exists here. The returned list from the prolog files contain the list of directions that the pacman needs to follow inorder to reach the goal.

3. dfs.P

This file contains the depthFirstSearch implementation in Prolog. The depth first search starts at depthFirstSearch(Start,Path,Direction). The arguments given to the method includes the start of the pacman state, the path that needs to be stored inside and the direction list that needs to be finally sent to the pacman gamestate. This file returns the list of the path along with the direction. The direction for each of the move is separated and is sent out as a list to the python file. This algorithm works by exploring the depth of the child first and then when the leaf node is reached it branches out. The leaf node is checked through the condition : depthFirstSearch1(Start,Path,Path,Direction) :- goal(Start). Once this consition is satisfied the prolog returns and explored the next states.

Sample output For the command : -l tinyMaze -p SearchAgent

goal goal(36).

[{'Path': '[8,14,20,26,32,33,34,28,29,30,36,[30,29,28,34,33,32,26,20,14,8],[29,28,34,33,32,26,20,14,8],[28,34,33,32,26,20,14,8],[34,33,32,26,20,14,8],[33,32,26,20,14,8],[32,26,20,14,8],[26,20,14,8],[20,14,8],[14,8],[8]]', 'Direction': '\_h201'}, {'Path': '[8,9,10,16,17,23,29,30,36,[30,29,23,17,16,10,9,8],[29,23,17,16,10,9,8],[23,17,16,10,9,8],[17,16,10,9,8],[16,10,9,8],[10,9,8],[9,8],[8]]', 'Direction': '\_h201'}]

connect(8,14,east).

8 14 east

connect(14,20,east).

14 20 east

connect(20,26,east).

20 26 east

connect(26,32,east).

26 32 east

connect(32,33,south).

32 33 south

connect(33,34,south).

33 34 south

connect(34,28,west).

34 28 west

connect(28,29,south).

28 29 south

connect(29,30,south).

29 30 south

connect(30,36,east).

30 36 east

connect(30,36,east).

36 east

connect(36,30,west).

4. bfs.P

This file contains the breadthFirstSearch implementation in Prolog. The breadth first search starts at breadthFirstSearch(Start,Path,Direction). The arguments given to the method includes the start of the pacman state, the path that needs to be stored inside and the direction list that needs to be finally sent to the pacman gamestate. This file returns the list of the path along with the direction. The direction for each of the move is separated and is sent out as a list to the python file. This algorithm works by exploring the breadth of the tree and then the child of each of the branched chidren. The leaf node is checked through the condition : breadthFirstSearch1(Q,[G,Start|Remaining],Direction) :- connect(Start,G,Direction), queue(\_,[Start|Remaining],Q,Direction),goal(G).

Sample output for the command : -l tinyMaze -p SearchAgent -a fn=bfs

?- breadthFirstSearch(8,Path,Direction).

Path = [35, 34, 28, 27, 21, 15, 14, 8],

5. AStar.P

This file contains the AStar algorithm implementation in Prolog. The AStar search starts at AStarAlgorithm(Start,Path,Direction). The arguments given to the method includes the start of the pacman state, the path that needs to be stored inside and the direction list that needs to be finally sent to the pacman gamestate. This file returns the list of the path along with the direction. The direction for each of the move is separated and is sent out as a list to the python file. This algorithm works by exploring the nodes using the heuristics. The heuristic is defined as the sum of the distance from the start and the distance of the node from the goal. The least distance is given the preference and it is selected for exploration. The leaf node is checked through the conditions : AStarAlgorithm1(Q,[G,Start|Remaining],Direction) :- connect(Start,G,Direction), queue(\_,[Start|Remaining],Q,Direction),goal(G).

6. CornersProblem.P

In the corners problem, we have tried to solve using the bfs method to eat all the four food present in the corners.