VIETNAM NATIONAL UNIVERSITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

PROJECT 01

ARTIFICIAL INTELLIGENCE

VÕ VĂN QUỐC HUY	18127113
VŨ CÔNG MINH	18127155
TỪ KIẾN VINH	18127255
TỪ KIẾN HOA	18127260

Ho Chi Minh City – 2020

TABLE OF CONTENTS

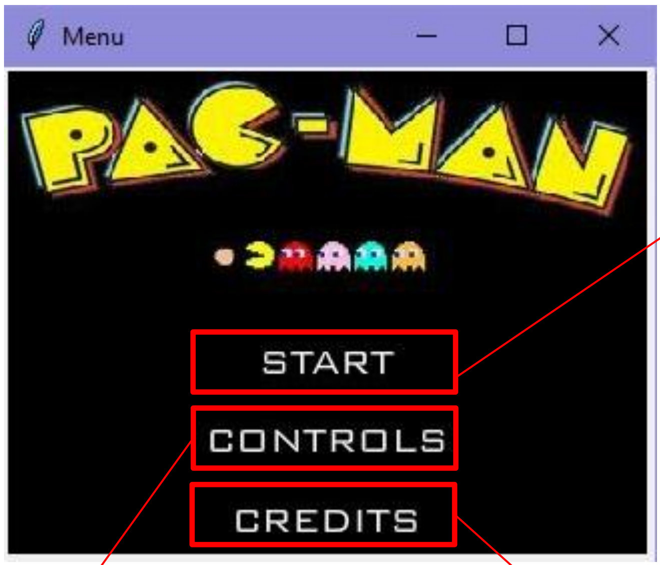
Assignment plan and overall

No.	Specification	Complete
1	Finish level 1 successfully.	✓
2	Finish level 2 successfully.	✓
3	Finish level 3 successfully.	✓
4	Finish level 4 successfully.	✓
5	Graphical demonstration of each step of the running process.	✓
6	Generate at least 5 maps with difference in number and structure of walls, monsters, and food.	✓
7	Report	✓

No.	Name_StudentID	Task
1	Võ Văn Quốc Huy_18127113	- BFS algorithm - Compare algorithms - Handle Input - Create Random Map
2	Vũ Công Minh_18127155 (Leader)	- Create and set up global variables, objects (Pacman, Monster, Food) - Handle windows (Start, control, main play) and widgets - Reverse_A_Star - Highest_Score_tactic - Blind_check_tactic
3	Từ Kiến Vinh_18127255	- Manhattan distance - A_Star algorithm - Monster chase tactic - Display score, tutorial
4	Từ Kiến Hoa_18127260	- Monster move arount init_pos - Nearest_food_tactic1 - Nearest_food_tactic2 - Monster move with animation - Pacman evade monster chasing

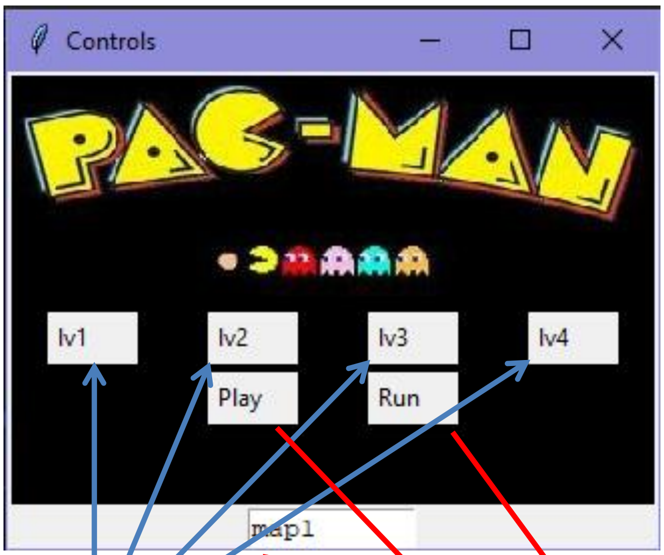
How to play

Start window

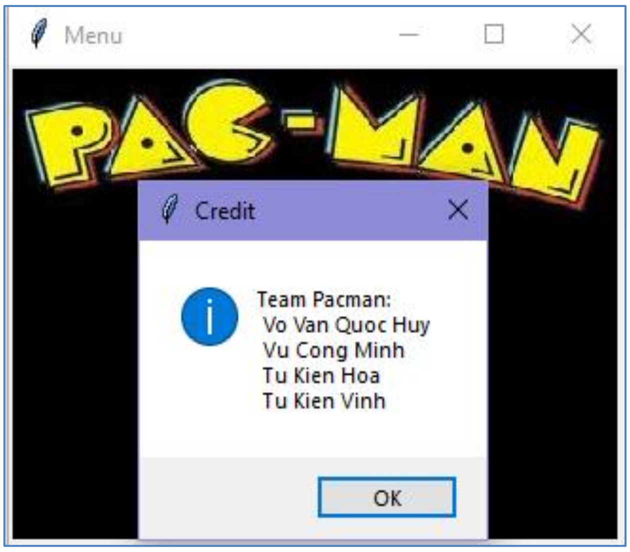


If you click START without setting from CONTROLS, it will be **level 1** and **map1.**

Control window



Credit window

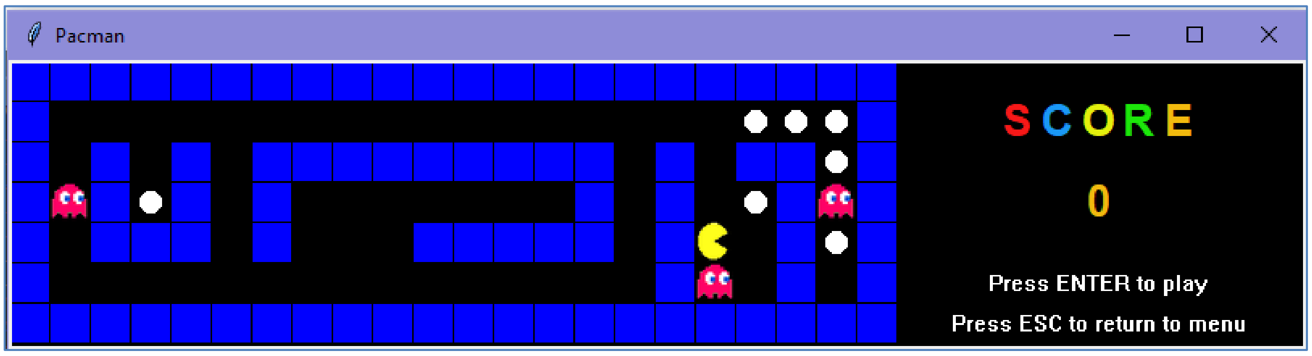


Team Pacman:
Vo Van Quoc Huy
Vu Cong Minh
Tu Kien Hoa
Tu Kien Vinh

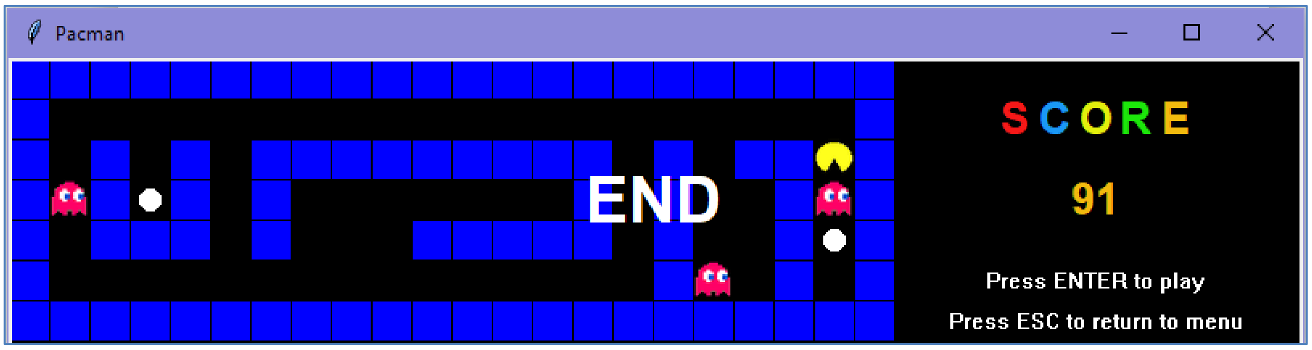
Set up level, input map (‘random’ or map name) and click “Play” or “Run” to return start window.

(Both of them will allow Pacman run automatically with algorithms because we haven’t built Play Mode for player vs computer)

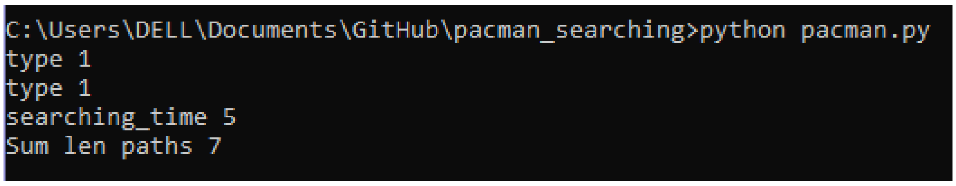
After click START, this is our main window to run algorithms:



Press Enter to watch Pacman running … and finished: (example at lv2)

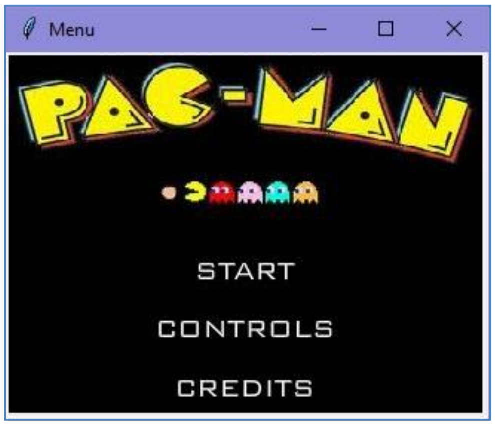


There are also some information printed out on console: (explain: there are **2 type1_ghost**, **Searching time** means the number of visited node when we call a searching algorithm, **Sum len path**s is the sum of path to each food)

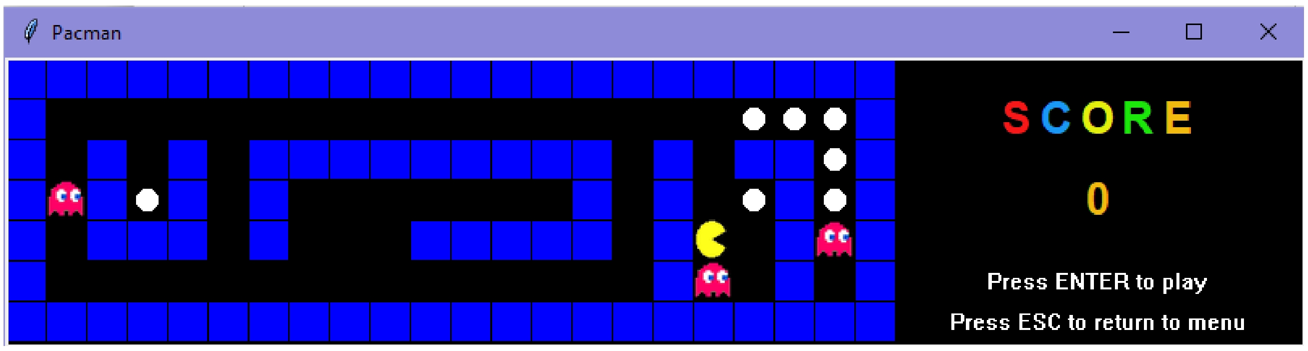


```
C:\Users\DELL\Documents\GitHub\pacman_searching>python pacman.py
type 1
type 1
searching_time 5
Sum len paths 7
```

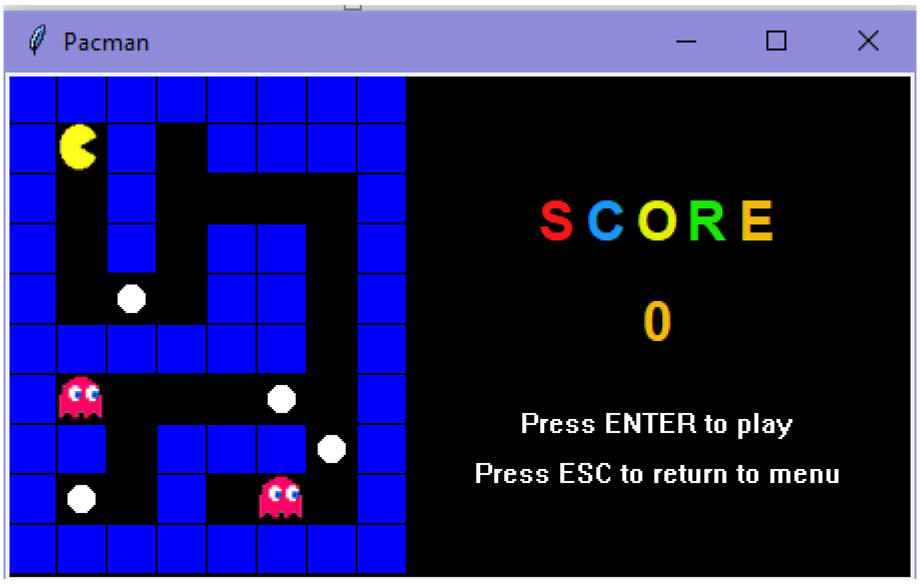
After all, press ESC to return menu (Start window):



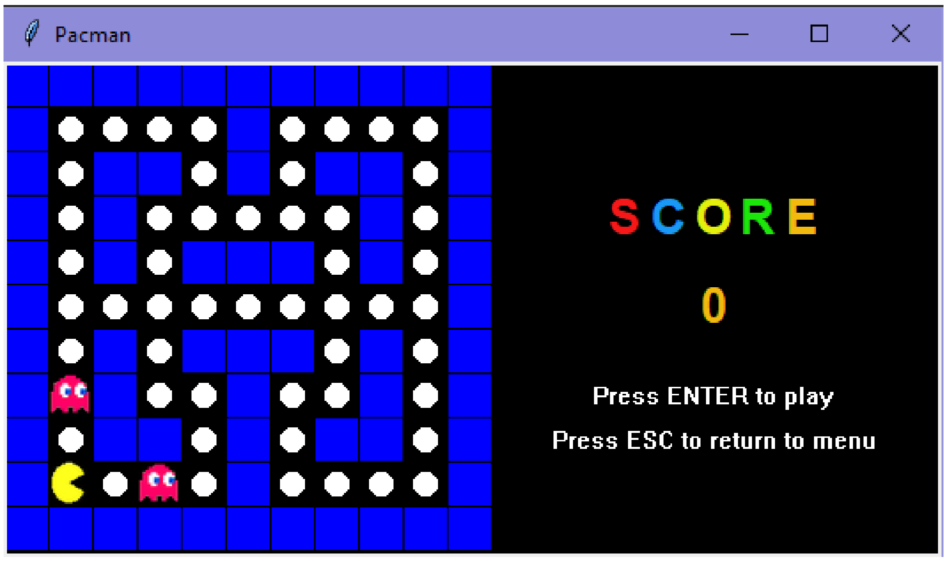
Map1:



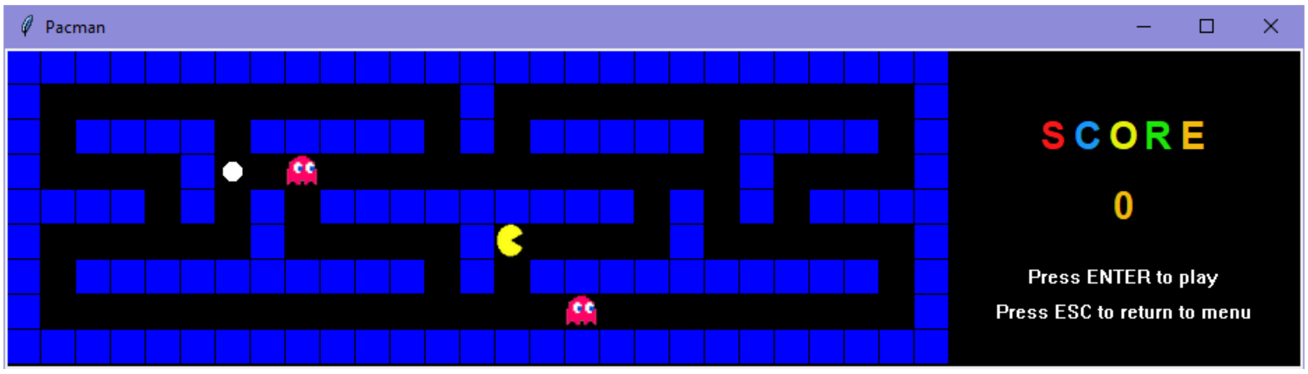
Map2:



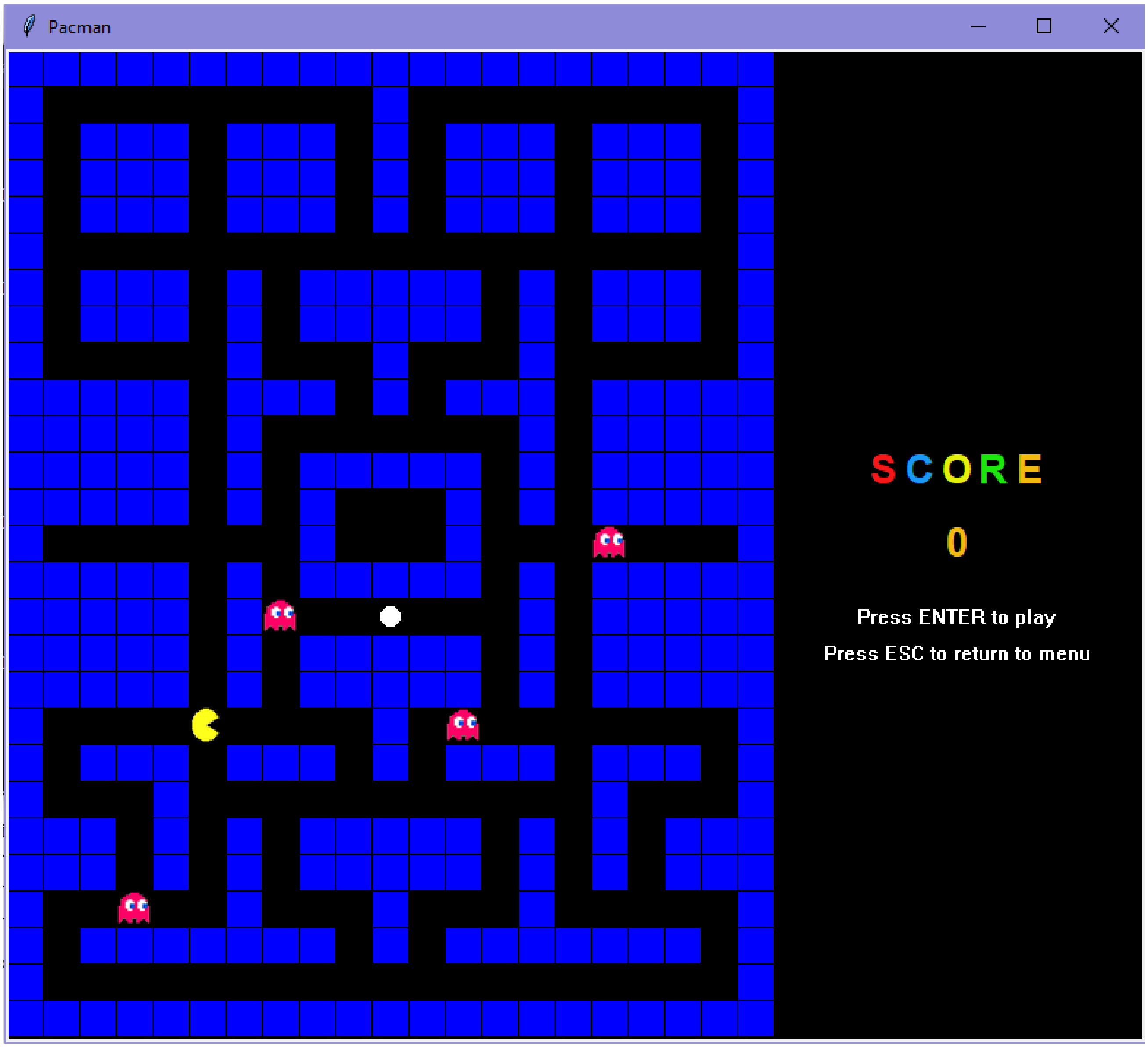
Map3:



Map4:


Pacman
SCORE
0
Press ENTER to play
Press ESC to return to menu

Map5:


Pacman
SCORE
0
Press ENTER to play
Press ESC to return to menu

Comparison between algorithms

1. Level 1:

Map 1	Searching time	Sum length paths
BFS	284	53
A*	87	53

Map 2	Searching time	Sum length paths
BFS	430	140
A*	240	140

Map 3	Searching time	Sum length paths
BFS	91	133
A*	153	133

Map 4	Searching time	Sum length paths
BFS	560	119
A*	295	119

Map 5	Searching time	Sum length paths
BFS	120	44
A*	195	44

2. Level 2:

Map 1	Searching time	Sum length paths
BFS	284	53
A*	134	46

Map 2	Searching time	Sum length paths
BFS	430	140
A*	229	140

Map 3	Searching time	Sum length paths
BFS	5	7
A*	154	7

Map 4	Searching time	Sum length paths
BFS	253	23
A*	53	25

Map 5	Searching time	Sum length paths
BFS	253	23
A*	189	25

At level 1,2: Opened map => use nearest_food_tactic1 function
We don't use DFS because it may not find optimal solution to the find food.

We choose A_Star because A_star is complete and optimal. A_star can expand fewer node than any other searching algorithms. With open_map, A* can be admissible if our heuristic function is good.

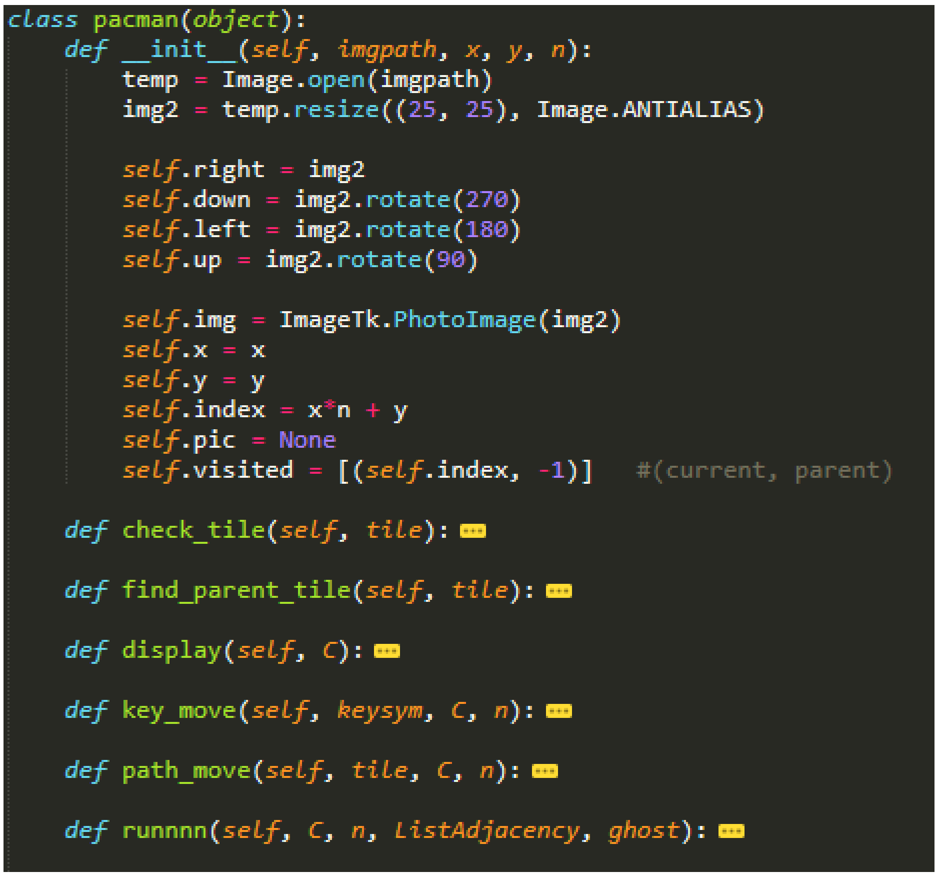
At level 3, 4: Closed map => use blind_check_tactic function

We choose DFS tree search because we can't evaluate the map and give an admissible heuristic function, so we have to travel and check each node. And with DFS, we can move back a step because DFS use stack to remember previous tile. Because of the number of steps, score is usually negative

Explaining code: objects.py

class pacman(object):

- Initialize pacman character from image file, resize to fit a block unit*unit (unit default is 25).
- Draw pacman at position x, y read from the map file.
- Each time the pacman moves, it erases the previous position and redraws it.
- Keep a visitedList for tracing back at lv3 and lv4. Each node is a tuple (current_tile, parent).
- There are two types of moving: follow key symbol ("Up Right Down Left") or move to a given index.
- Function runnnn evade monster by random a move in its AdjacencyList which helps increase distance between pacman and monster.



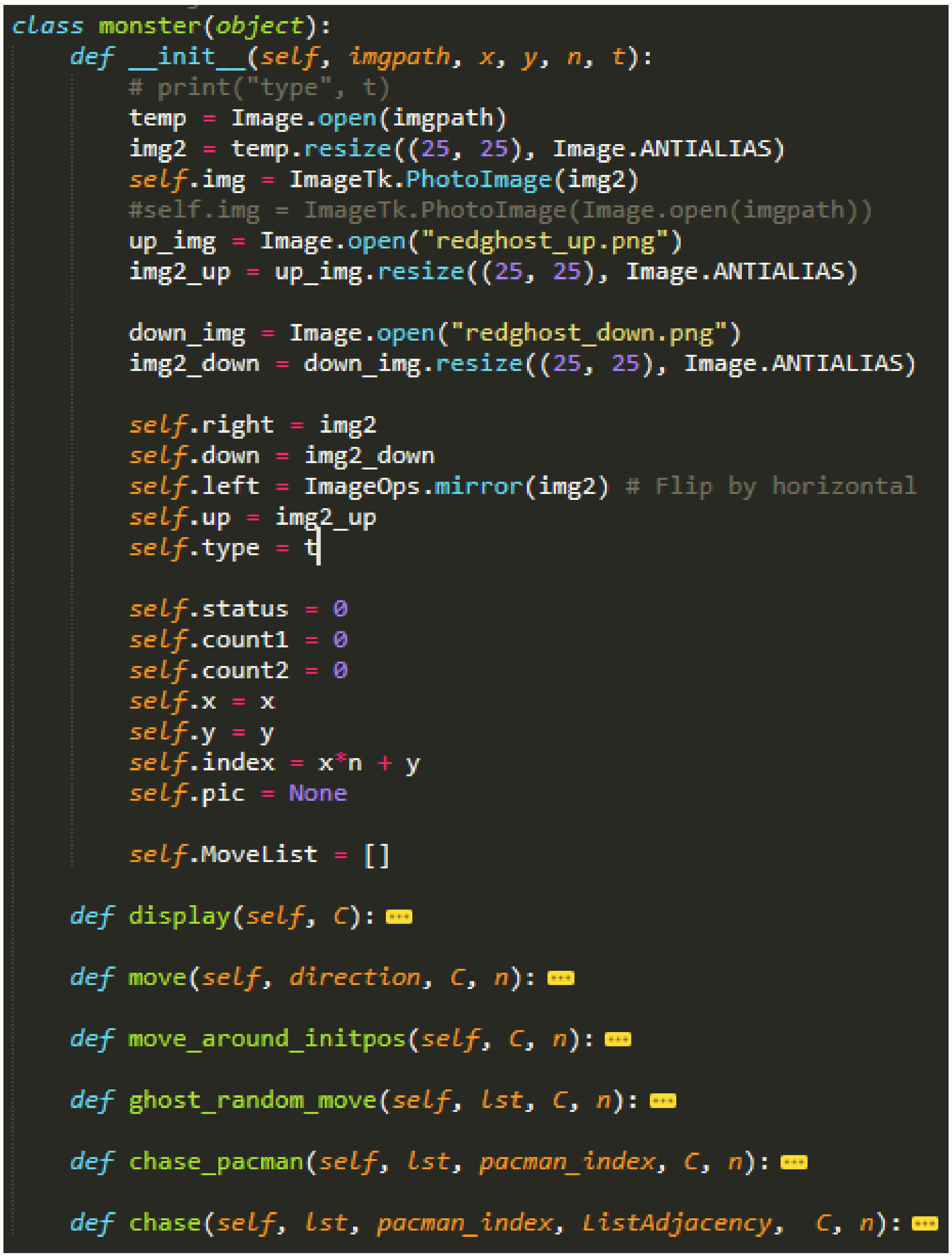
```
class pacman(object):
    def __init__(self, imgpath, x, y, n):
        temp = Image.open(imgpath)
        img2 = temp.resize((25, 25), Image.ANTIALIAS)

        self.right = img2
        self.down = img2.rotate(270)
        self.left = img2.rotate(180)
        self.up = img2.rotate(90)

        self.img = ImageTk.PhotoImage(img2)
        self.x = x
        self.y = y
        self.index = x*n + y
        self.pic = None
        self.visited = [(self.index, -1)]   #(current, parent)

    def check_tile(self, tile): ...

    def find_parent_tile(self, tile): ...

    def display(self, C): ...

    def key_move(self, keysym, C, n): ...

    def path_move(self, tile, C, n): ...

    def runnnn(self, C, n, ListAdjacency, ghost): ...
```

class food(object):

- Initialize the cells with the value Food as a white circle.
- If the cells are eaten, they will be deleted by themselves.

class monster(object):

- Generate from Monster image file and draw in place on maze.
- Like the pacman, each time Monster moves will be deleted and redrawn in a new position.
- Will initialize random positions from array of positions and Monster will randomly move.
- Chase the pacman calculated according to the distance closest to the pacman.
- In lv3: Monster move around initial location (9 tiles available)
- In lv4: Monster is chosen a chase way with its random type.
  + Type 0: 3 steps chasing -> 5 steps random move -> 3 steps chasing …
  + Type 1: If manhattan distance between it and pacman > 5 then random move, else chasing.



```python
class monster(object):
    def __init__(self, imgpath, x, y, n, t):
        # print("type", t)
        temp = Image.open(imgpath)
        img2 = temp.resize((25, 25), Image.ANTIALIAS)
        self.img = ImageTk.PhotoImage(img2)
        #self.img = ImageTk.PhotoImage(Image.open(imgpath))
        up_img = Image.open("redghost_up.png")
        img2_up = up_img.resize((25, 25), Image.ANTIALIAS)

        down_img = Image.open("redghost_down.png")
        img2_down = down_img.resize((25, 25), Image.ANTIALIAS)

        self.right = img2
        self.down = img2_down
        self.left = ImageOps.mirror(img2) # Flip by horizontal
        self.up = img2_up
        self.type = t

        self.status = 0
        self.count1 = 0
        self.count2 = 0
        self.x = x
        self.y = y
        self.index = x*n + y
        self.pic = None

        self.MoveList = []

    def display(self, C): ...

    def move(self, direction, C, n): ...

    def move_around_initpos(self, C, n): ...

    def ghost_random_move(self, lst, C, n): ...

    def chase_pacman(self, lst, pacman_index, C, n): ...

    def chase(self, lst, pacman_index, ListAdjacency,  C, n): ...
```

Explaining code: pacman.py

Function random_Maze():

- Randomly initialize two variables for map size, then create a random matrix with three values of 0, 1, 2 and then surround the matrix with value 1.
- Then random the number and value of Ghost position corresponding to a random position in the matrix.
- Assign matrix into array corresponding to each row of matrix and return the results.

Function handle_input():

- Initialize the variable **lst** as an array, open the file containing the position and matrix.
- For each line read, it is detached and fed into **lst**

Function create_maze(C):

- For each input box from the matrix will be considered at that position as Wall or Food or Ghost.
- For each position there will be additional functions.
- Display all objects on canvas.

Function create_data(C):

- If level = 2 then converts the positions with Monster value to 1. Else its value is 0 because monster's initial location is viewed as road(visitable).
- Go through the maze array, if it is a sugar or Food value, enter the **ListAdjacency** matrix. Each element in ListAdjacency is a list of tuples **(node_adjacency, node_properties).** If current node is wall, ListAdjacency of it will be [None]
- Create List of moveable tiles for each ghost for random around initial location function.

Function display_score():

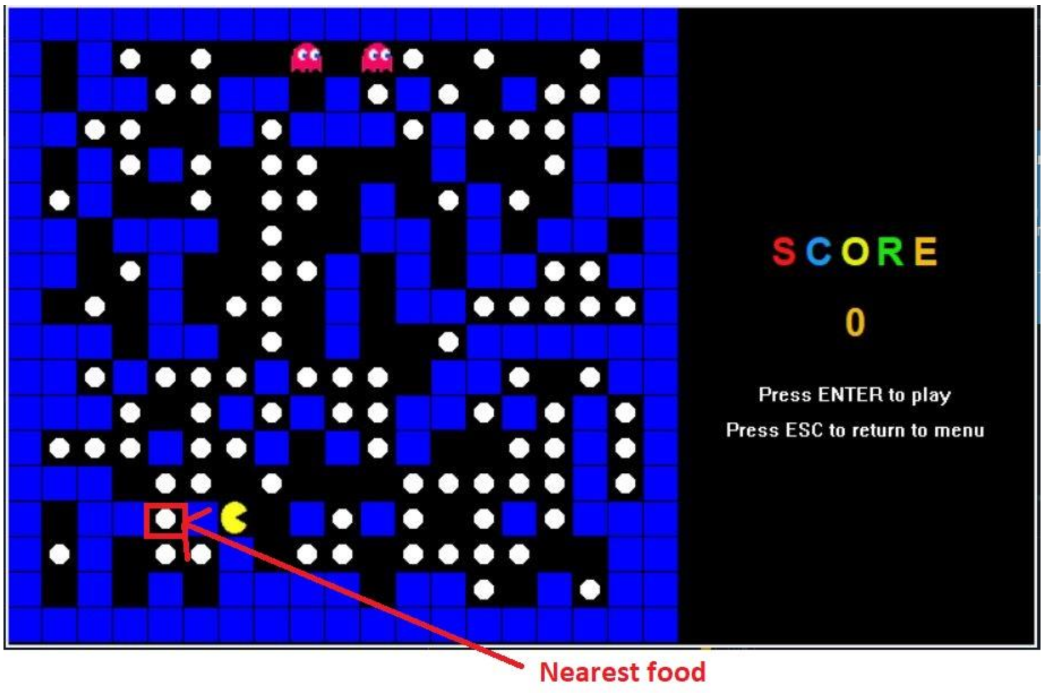
- Draw a score board

Function sort_Food():

- Sort ListFood by ascending distance from pacman location.

Function nearest_food_tactic1():

- Iterate each food object in **ListFood**.
- While loop for **ListFood** (While len(**ListFood**) > 0):
- Sort food objects in ListFood by **sort_Food()** function.
- Calculate path from Pacman's current position to ListFood[0] (food that has nearest distance calculated by Manhattan distance between pacman's position and food's position) using A* search, **if path not found or the path to food is too far** (cost more than 20 while the score of food is only 20), remove the current food from ListFood.
- Pacman moves one step (follow path) and decrease score by 1 (each move cost 1 score), check if the new current position has food or not, if there is a food in current position, increase current score by 20 and delete the food Pacman ate from the

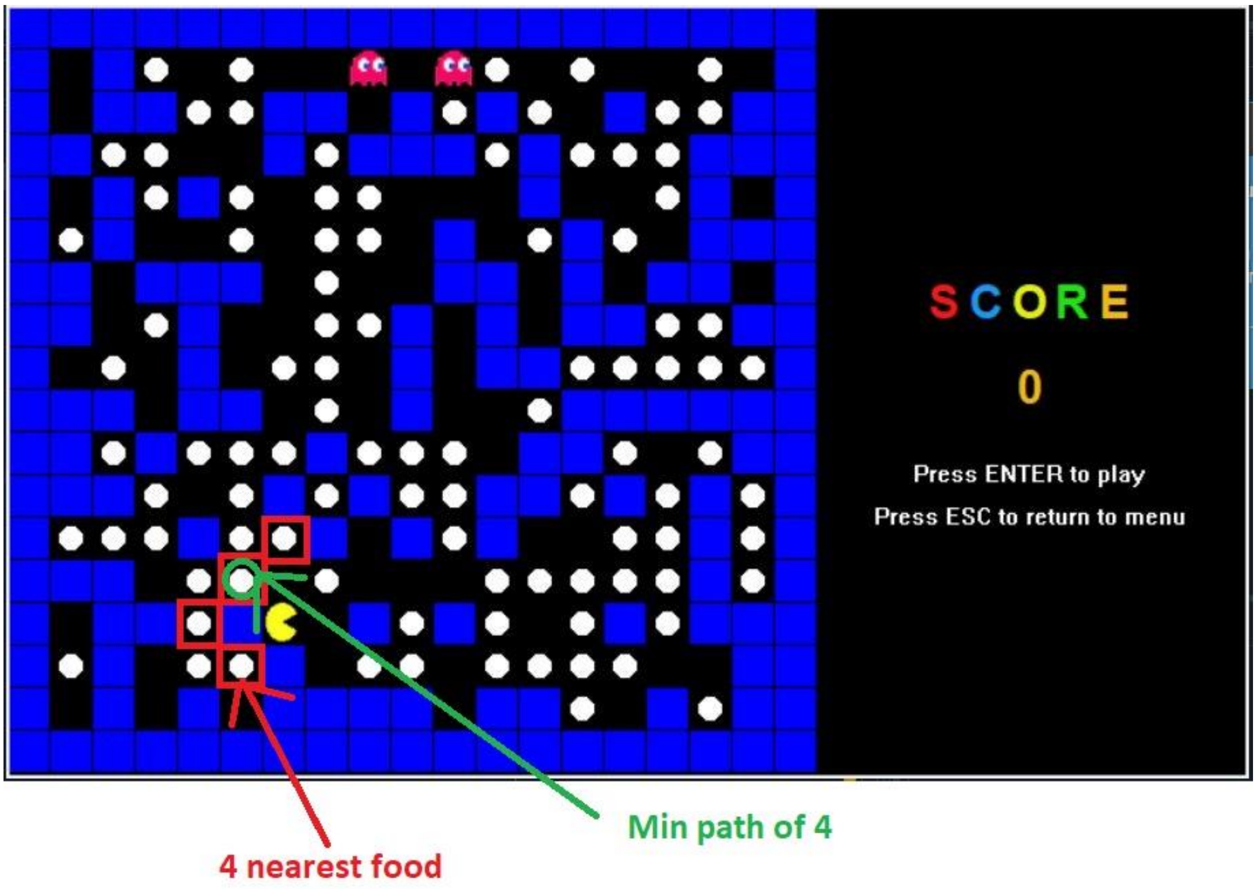


Disadvantage:

- Pacman is stuck (move around 2 tiles) in some cases that there are 2 food nearby. For example, there are 2 food A and B, Pacman has the path to food A, after go 1 step in the path to A, the nearest food is B so Pacman will follow the path to B (calculated after go 1 step in path to A). After moving one step in path to B, the nearest food is A so the path to A is calculated and Pacman follows this path, this make Pacman stuck in 2 tiles.
- Miss some food because some food that can't reach at the current state is removed from **ListFood** (include food that is blocked by monsters and food that can't reach). Therefore, in the next state if the monsters move far away from food that they blocked in the previous state, Pacman still ignores these food.

Function nearest_food_tactic2():

- Upgrade from **nearest_food_tactic1()**
- Iterate each food object in **ListFood**.
- While loop for **ListFood** (While len(**ListFood**) > 0):
- Sort food objects in **ListFood** by **sort_Food()** function.
- Create a copy of **ListFood** (**ListFood_currentState**) to temporarily remove some food that Pacman can't reach with current position (the food is blocked by ghosts) out of copy of **ListFood**, these removed food objects in the copy list is still in **ListFood** so in the next loop we can recheck if it is possible to reach them or not.
- Calculate path from Pacman's current position to **ListFood_currentState[0]** (food that has nearest distance calculated by Manhattan distance between pacman's position and food's position) using **A* search**. While the path is not found, remove first food in **ListFood_currentState** and calculate path of first food of new **ListFood_currentState** until the path is found or there is no food in list.
- If there are at least 4 food objects in the list, calculate path to each of 3 food after, assign shortest path to **path**.
- While loop to move follow the path:
- Check if Pacman's current position is near any ghost or not, if there is a ghost nearby (calculated distance by Manhattan distance, distance <= 2), Pacman will find the possible direction to evade the ghost (use **runnnn** function), if there is no possible move, Pacman stays at current position. Break the while loop of path if Pacman evaded the ghosts, move to next loop and calculate new path.
- Pacman moves one step (follow **path**) and decrease score by 1 (each move cost 1 score), check if the new current position has food or not, if there is a food in current position, increase current score by 20 and delete the food Pacman ate from the map.

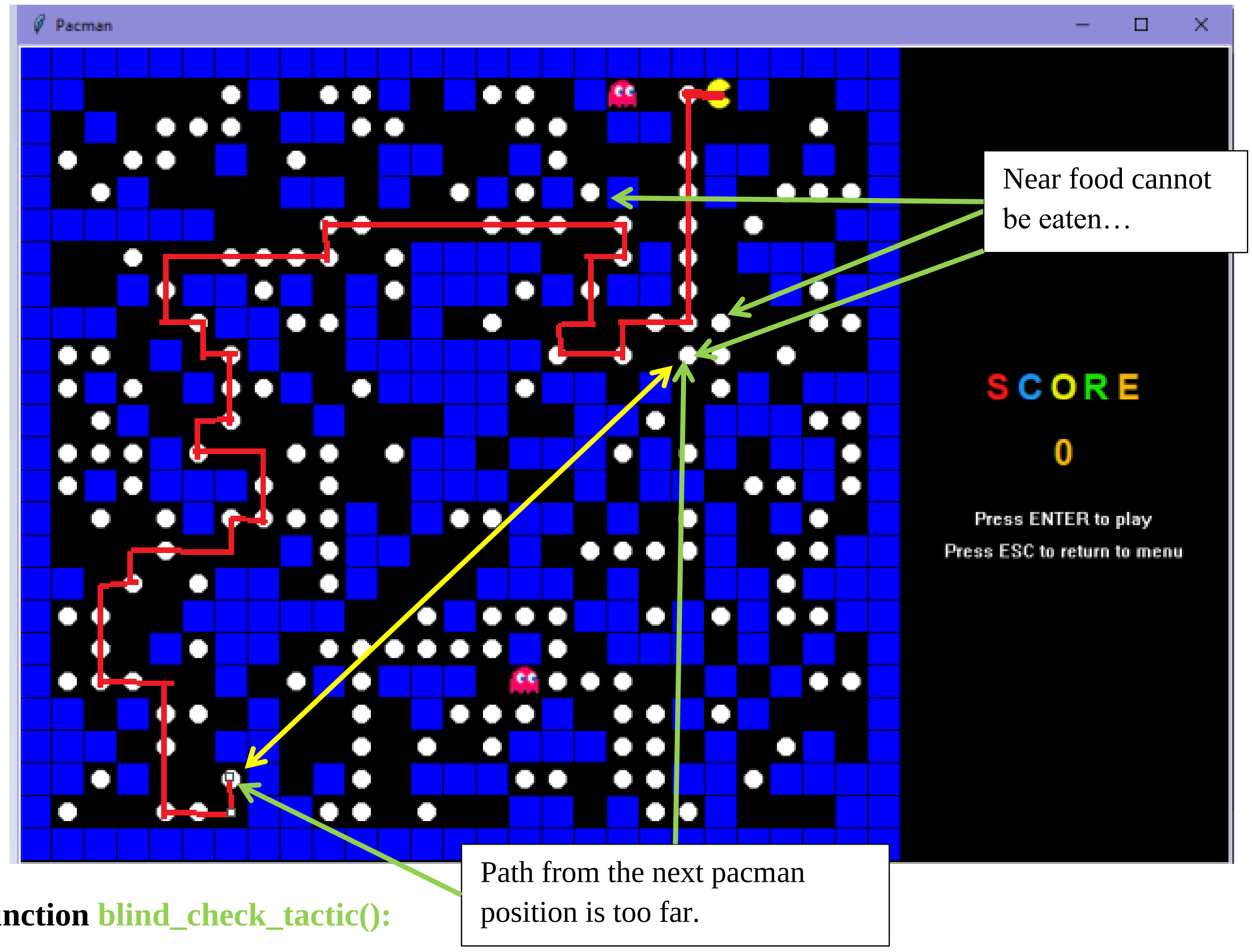


Function **highest_cost_tactic():**

- Use **reverse_A_Star** algorithm for each food in **ListFood** to create a **ListCost**
- Go to food which has highest score in its path (visit the most food in the path).
- When pacman reach the goal, use **reverse_A_Star** again to find from the rest of ListFood.
- **Disadvantage**:

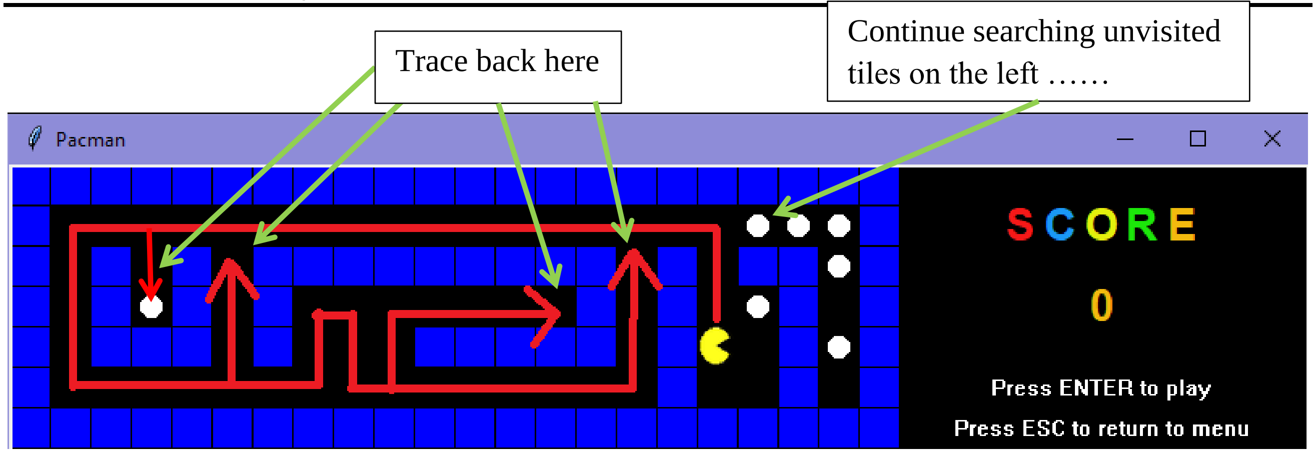
  + We have to use **reverse_A_Star** for all food in ListFood, run pacman then update again and again. It cost very long time for searching.

  + This isn't an optimal tactic because some food near the path can't be eaten when pacman running. And then, when pacman reach goal, the distance from it to pacman is further, and reverse_A_Star will search it again.


Pacman
SCORE
0
Press ENTER to play
Press ESC to return to menu
Near food cannot be eaten…
Path from the next pacman position is too far.

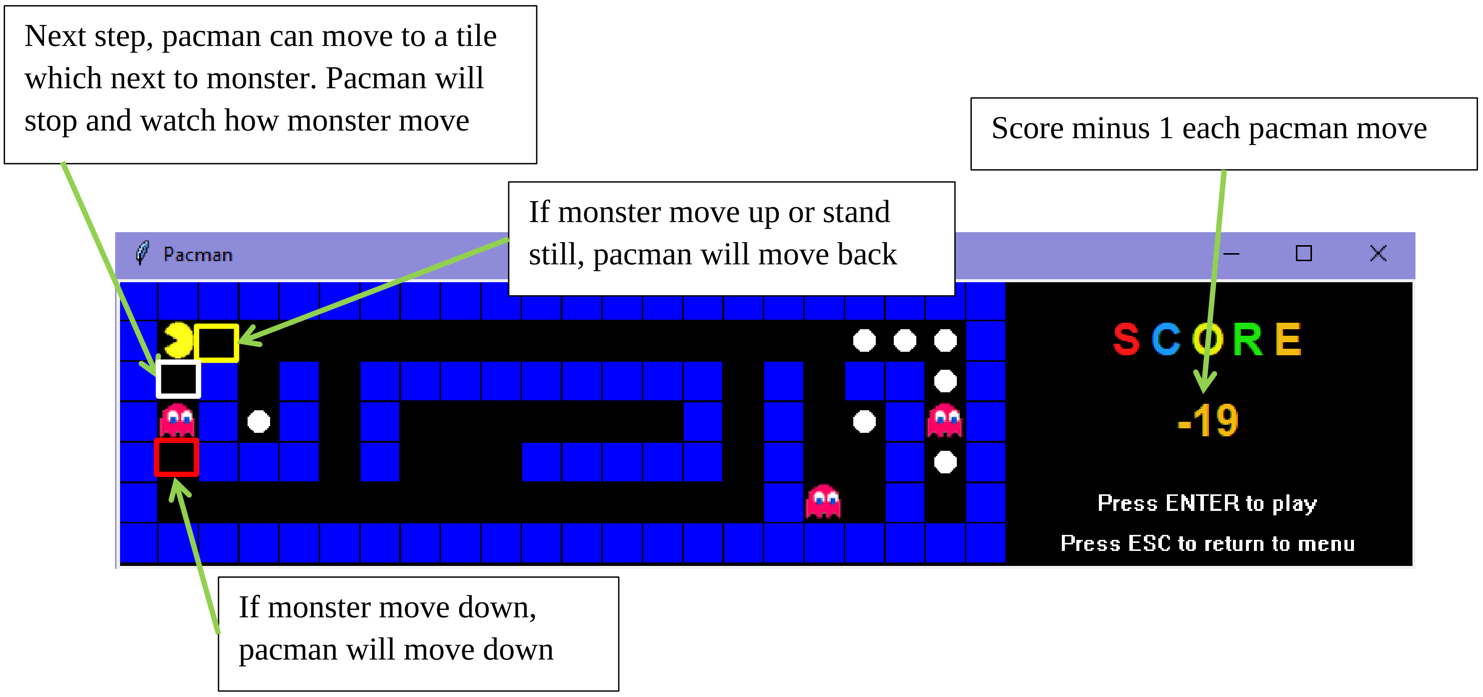
Function **blind_check_tactic():**

- Use **Stack** to iterate like DFS tree searching
- Find **unvisited tile** in **ListAdjacency[pacman_index]** and that tile **doesn't near monster** then visit it, push it into **Stack.**
- If **check_ghost_1** found that tile near monster (tile in ListAdjacncy[monster_index] or manhattan distance between pacman and monster is 2), pacman wait a step, watch monster move.
- After monster_move, **check_ghost_2** again, if that tile still near monster, don't move to that tile, choose another one.
- If we have checked all tiles in **ListAdjacency[pacman_index],** move back one step at the top (last) of Stack.



Trace back here

Continue searching unvisited tiles on the left ……



Next step, pacman can move to a tile which next to monster. Pacman will stop and watch how monster move

Score minus 1 each pacman move

If monster move up or stand still, pacman will move back

If monster move down, pacman will move down

Function Play():

- Main function, **declare global variables**: lst, ListGhost, ListAdjacency, ListFood, data, tkinter top, canvas C, pacman P, …
- Display **SCORE, introduction,** …
- Call **create_maze(), create_data(),…**

Other functions: handle **button clicked** or **key pressed,** display **graphic,** create **setting window, message up team information …**

Explaining code: Searching_Algorithm.py

Function BFS (adjacency_list, begin, food_position):

- Create the parent array the size of the passed list.
- Check at start is Food or not, if so then return results.
- Create the queue for the starting value, node at the position taken from the queue. Check to see if it is in the list expanded or not, if the node at location is Food, retrieve it and return the function values.

Function get_manhattan_heuristic(current_position, food_position, maze_size):

- Get position and Food position.
- Then calculate the distance between two positions and return the result.

Function A_Star(adjacency_list , current_position, food_position, maze_size):

- Initialize three arrays: explored, frontier, path
- Initialize the parent array to -1.
- Each node pops out with 2 values, its cost and index. We calculate the fcost and the hcost and then we have the gcost.
- Check if that Food location has the same node value.
- If not, check if that node has been expanded, if not, then enter and update the frontier and parent again.

Function reverse_A_Star(adjacency_list, current_position, food_position, maze_size, ghost_list, food_list):

- Based on A* searching **using heuristic function.**
- We don't use min_cost node in frontier to expand. We'll **expand max_cost node**.
- If node is a food, add 20 to cost, else minus 1 to cost.
- Return: **len(visitedList), visitedList, path, fcost** with fcost is the cost of goal node.

References

Link github:

https://github.com/MinhVu2018/pacman_searching

Algorithms:

Based on pseudo code from lectures.

Libraries:

tkinter Python: create windows.

PIL Python: handle image objects.

Time, Random library: Pause game, use random function

Numpy library: Create random maze easily

Pacman Game Graphic:

http://effbot.org/tkinterbook/place.htm

https://github.com/jeremylowery/slideshow

https://effbot.org/tkinterbook/canvas.htm

Images from google and transparent by using: https://onlinepngtools.com/create-transparent-png