PROJECT 01: SEARCH

Information about us:

No.	Full name	Student ID
1	Nguyễn Hoàng Nhân	18127017
2	Kiều Công Hậu	18127259
3	Trần Đình Sang	18127267
4	Trần Thanh Tâm	18127268

PACMAN

1. Assignment plan:

No.	Task	Implement
1	Design architecture and algorithms. (Ex: Pacman, Monster, Graphics …)	Whole team
2	Design algorithm for each level	Nguyễn Hoàng Nhân Kiều Công Hậu Trần Thanh Tâm
3	Design graphics	Trần Đinh Sang
4	Design map	Trần Đình Sang
5	Initial map	Trần Thanh Tâm
6	Greedy Search A* (Level 1 – Level 2 for Pacman, Level 4 for Monster)	Kiều Công Hậu
7	Heuristic Local Search using DLS (Level 3 – Level 4 for Pacman)	Nguyễn Hoàng Nhân Trần Thanh Tâm
8	Peas Tracking (Level 3 – Level 4 for Pacman)	Nguyễn Hoàng Nhân Kiều Công Hậu
9	Report	Whole team
10	Test	Trần Đình Sang

2. Environment:

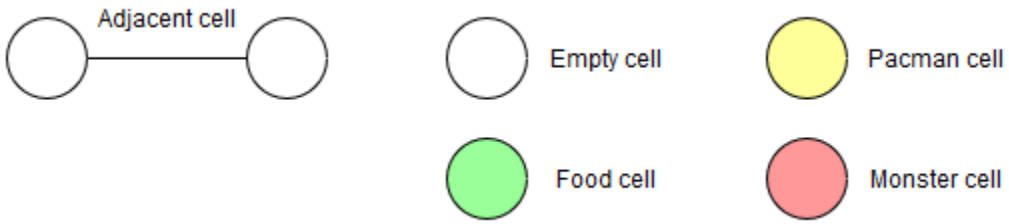
– Python 3.7
– Graphics: Pygame

3. Self-assessment:

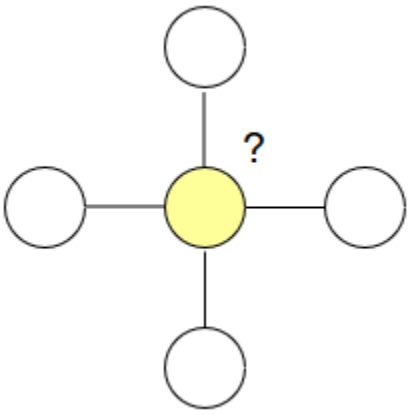
No.	Requirement	Note	Completement
1	Level 1: Pac-man know the food's position in map and monsters do not appear in map. There is only one food in the map.	We use Greedy Search A* to find the shortest path to the food.	100%
2	Level 2: monsters stand in the place ever (never move around). If Pac-man passthrough the monster or vice versa, game is over. There is still one food in the map and Pac-man know its position.	Same to level 1. But in this level, we consider monster as a wall, then find the shortest path to food. If monster blocks the way to food that means Greedy Search A* have no results hence Pacman loses.	100%
3	Level 3: Pac-man cannot see the foods if they are outside Pacman's nearest three-step. It means that Pac-man just only scan all the adjacent him (8 tiles x 3). There are many foods in the map. Monsters just move one step in any valid direction (if any) around the initial location at the start of the game. Each step Pacman go, each step Monsters move.	In this level, we use which we call Heuristic Local Search and Peanut Tracking. We will fully descript later.	100%
4	Level 4 (difficult): map is closed. Monsters will seek and kill Pac-man. Pac-man want to get food as much as possible. Pacman will die if at least one monster passes him. It is ok for monsters go through each other. Each step Pacman go, each step Monsters move. The food is so many	In this level, we use Greedy Search A* for each monster move to repeatedly find the shortest path to Pacman then perform only one move.	100%
5	Graphical demonstration of each step of the running process		100%
6	Generate at least 5 maps with difference in number and structure of walls, monsters, and food.		100%
7	Report your algorithm, experiment with some reflection or comments.		100%
Total			100%

4. Discussion:
    a. Heuristic Local Search:

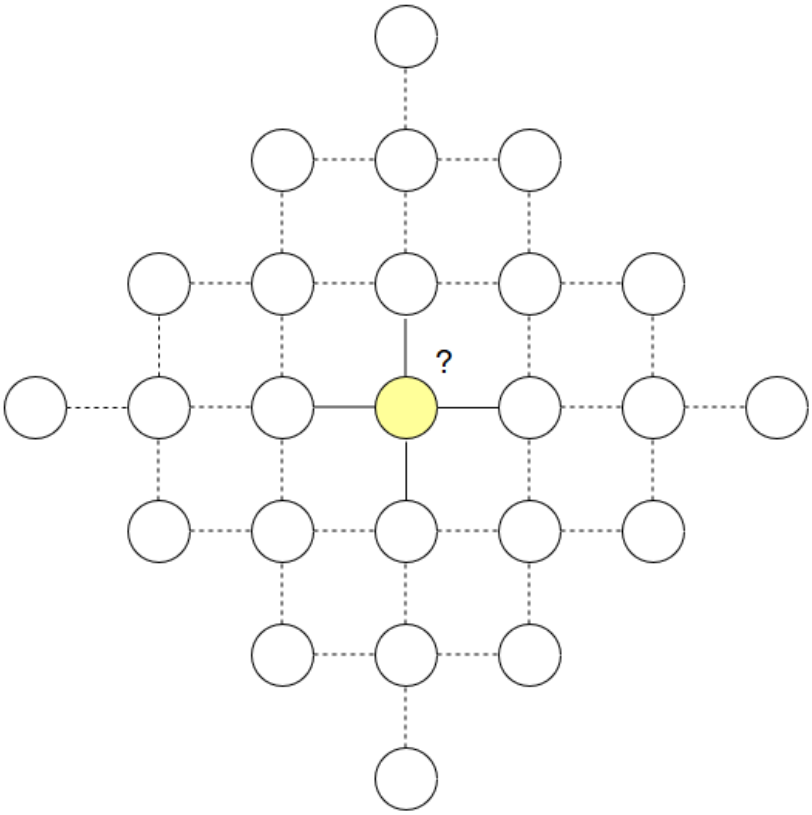
First, we have some symbol define:



Local search decide which cell Pacman should move next.



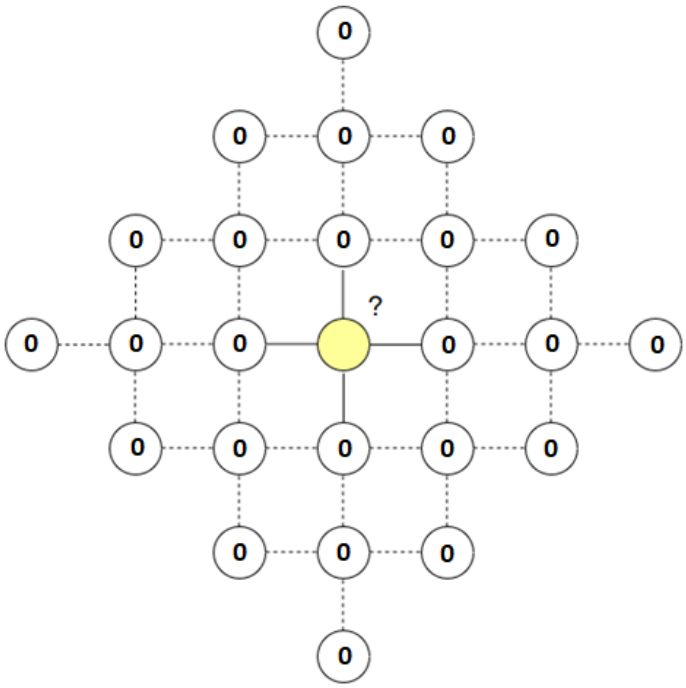
In the visual of Pacman (8 tiles * 3) in our understanding is **3 level of Depth Limited Search** from the Pacman's cell, so we assume the visual of Pacman like this:



Every cell has 2 main attributes:

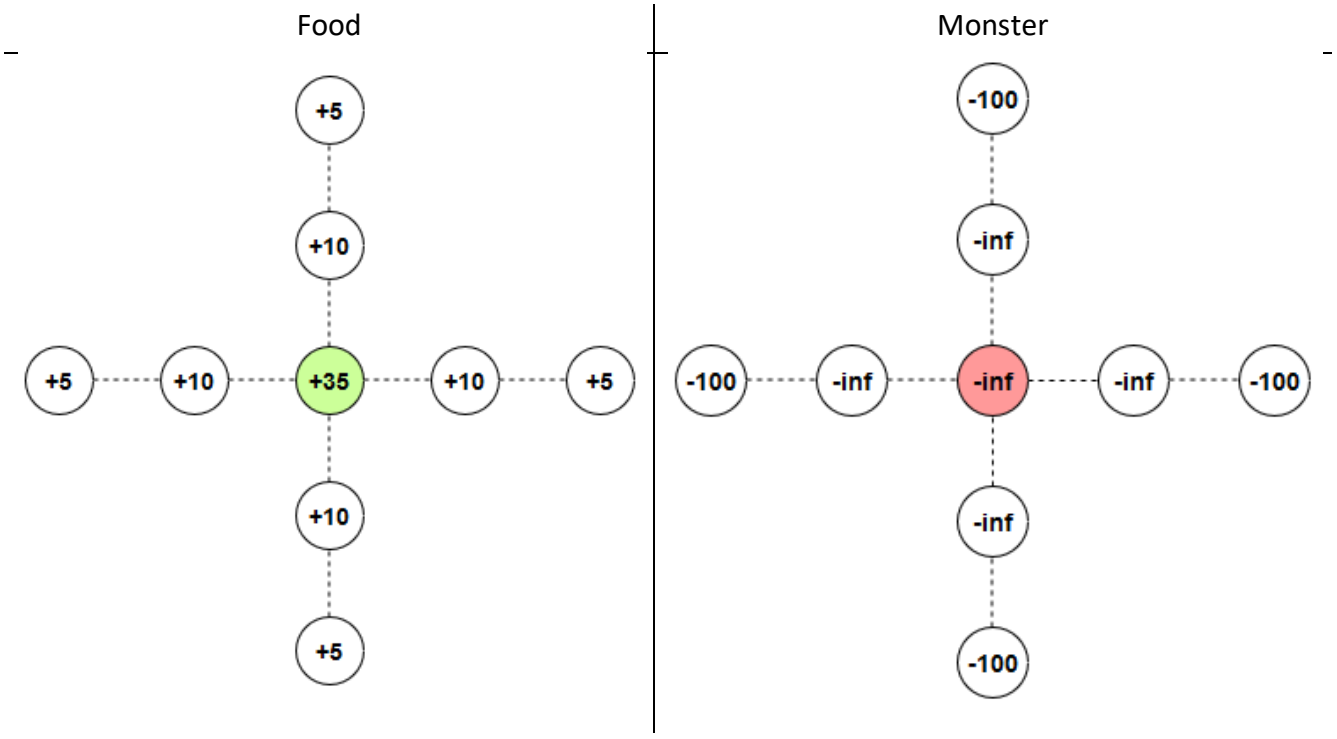
- **Heuristic:** initialize with 0, reset to 0 for every time before the heuristic function runs.
- **Visited:** initialize with 0 and count the number of times that Pacman pass the cell.

To help Pacman decide which cell to move next, we run an algorithm to find the heuristic of each cell that in visual of Pacman. But first, we reset heuristic to 0 at all cell in Pacman's visual by using DLS travel 3 level.



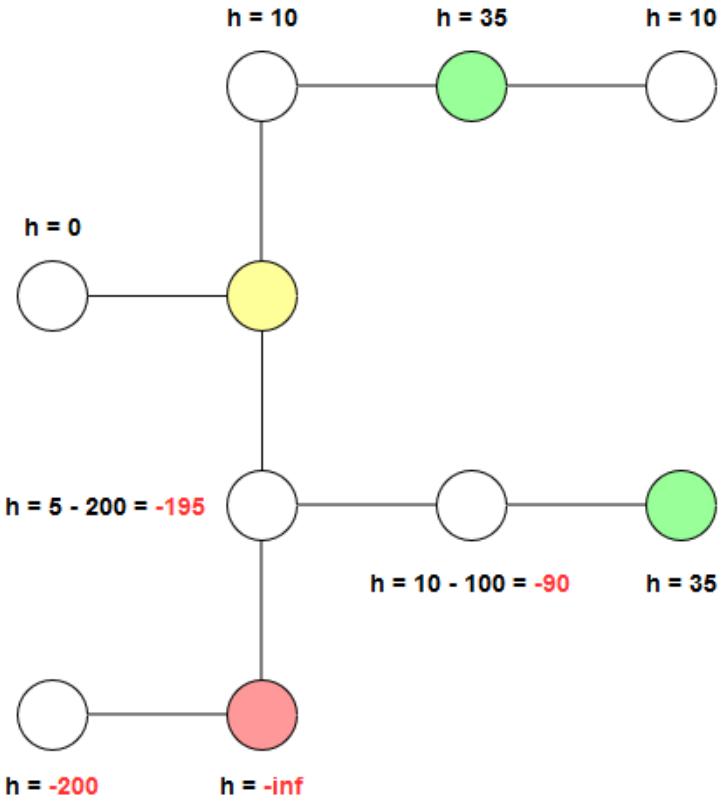
Then, we use DLS to travel 3 level of depth in Pacman's visual (main). For each cell that DLS travel:

- Step 1: Check the current cell is Food or Monster go to step 2, else continue DLS (main).
- Step 2: Use DLS (sub) to travel 2 level of adjacent cell to the current cell to update the heuristic following the rule below but if the DLS (sub) travel to Pacman's cell then it will stop.

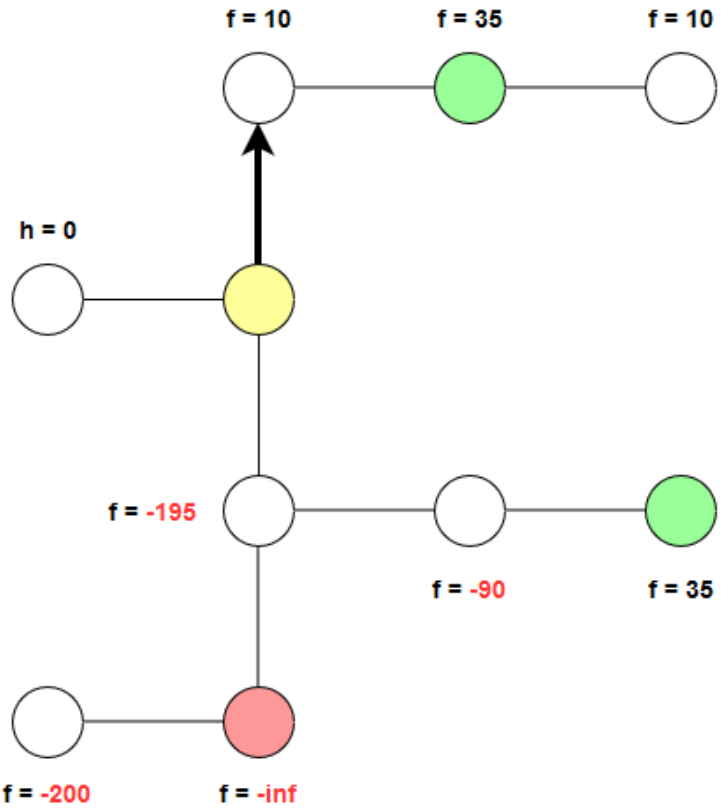


- Step 3: Continue DLS (main).

For example:



After calculate heuristic for all cell in Pacman's visual, we use Local Search to find $Max_{f(n)}$ of all adjacent cell to Pacman's cell for $f(n)=h(n)-v(n)$ with $h(n)$ is heuristic and $v(n)$ is visited.



(in this example, for simplistic $v(n)=0$ for all cell)

b. Peas Tracking:

– General idea:

When Pacman discovers a new food, it needs to remember the position where it discovers that food (which is it's current position). Because if it doesn't go to eat that food instantly but go to another direction instead, it'll still remember the existence of that food and can trace back to where it found that food, and go to eat it. This will assure that no food will be forgotten as long as they have been seen by Pacman.

When Pacman finds a new food, it'll start dropping peas from its current position, and dropping along whichever path it will go next (regardless whether it will go to eat that food or go to other direction). Different food will be managed by different type of peas (peas with different color, for example). Pacman will keep dropping peas until it stops seeing any food. Then, it will stop dropping peas and start backtracking using the peas it dropped. It backtracks to the position where Pacman discovered the last new food. While backtracking, it picks up all peas on its way. And then it continues the loop of dropping peas and backtracking again.

Note 1: When a food is eaten, all peas keeping track of this food will be ignored and forgotten.

Note 2: After a backtracking process finishes, the remaining peas which haven't been picked up are still remembered by Pacman, and eventually they will be picked up or be forgotten when the food it's managing has been eaten.

Note 3: When Pacman discovers a new food (we'll call it food A for example), he'll start dropping peas to keep track of this food. Pacman decides not to eat food A right away, but go to another direction to eat other food first. So at some point, he may lose sight of food A. Later on, Pacman suddenly sees food A again, but this time its current location is different from where he discovered food A for the first time. In this case, we'll consider Pacman's current location to be the new discovery location of food A. That means Pacman will discard all peas of food A he has dropped until now, and starts dropping new peas from this location. This will help reduce the length of the backtracking path which Pacman has to go.

Note 4: When Pacman discovers a new food, but there's a monster near that food, Pacman will give up on that food. That means Pacman will ignore and forget that food, while the heuristic algorithm will guide Pacman to go another direction, thus making sure Pacman will not try to come back to eat that food. This only happens in Level 3, when the monster only move around its initial location, making it difficult for Pacman to eat the food in this situation. On Level 4, the monster can move anywhere freely, thus coincidentally make it feasible for Pacman to eat the food in this kind of situation.

– Implementation:

In Python, we implement this idea by using list and stack data structure. We create a list named "peas" (for example) to keep track of all kinds of peas the Pacman has dropped. In list "peas", there are many sub-list, each sub-list manages one type of peas used to track one food. In each sub-list, there are 2 components: one is a tuple storing the coordinate of the food it's managing, and the other is a stack managing the coordinates of all position where the peas of this type have been dropped (we'll call this list the "path" of this type of peas).

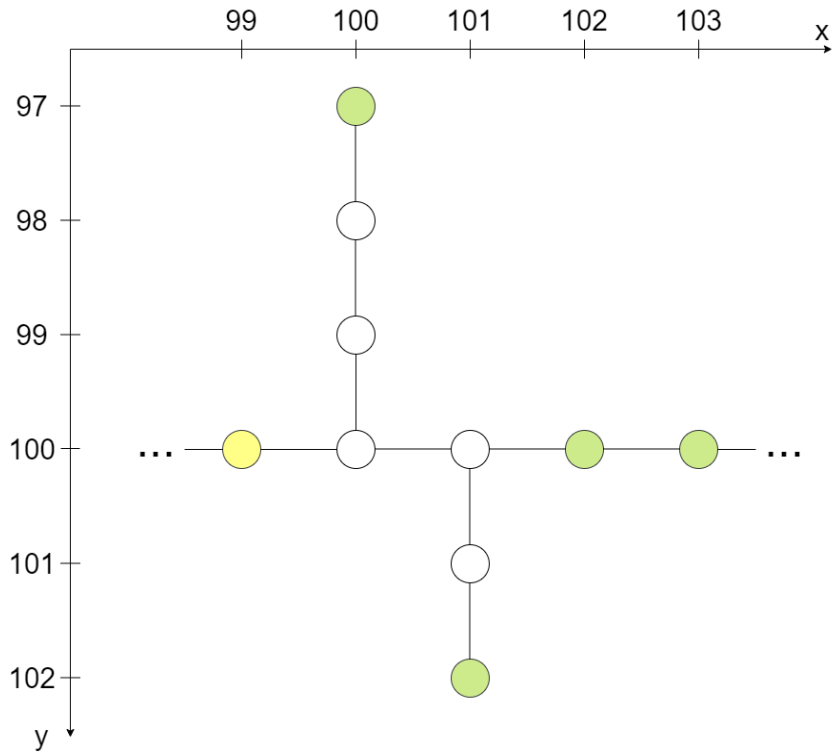
When Pacman finds a new food, a new sub-list will be created in list "peas", containing the coordinate of that food and an empty "path" stack. For every move Pacman makes, the coordinate of the position Pacman has just left will be push to the "path" stack in **every** sub-list of list "peas" (this is equivalent to the Pacman's action of dropping peas). When a food is eaten, the sub-list in list "peas" which manages that food will be removed. When Pacman re-discovers an old food (see Note 3 in the General idea above), the "path" stack associated with that

food will be reset (i.e. all its data will be deleted) and new coordinates will be pushed in from that point onwards. When Pacman discovers a new food, but there's a monster near that food (see Note 4 in the General idea above), the sub-list associated with that food will be discarded, while the heuristic algorithm will guide Pacman to go another direction, thus making sure the food won't re-create a new sub-list in list "peas" again.

When Pacman stops seeing any foods, it'll start backtracking. Pacman will choose to go along the path stored in the **shortest** "path" stack (this means Pacman will backtrack to the position where it discovered the last new food). At each position, it'll pop the element at the top of **every** "path" stack (these elements are the same) to get the next position it needs to go.

After the backtracking process finishes, Pacman will continue the cycle of dropping peas and backtracking again and again.

– Example:



Initially:

Pacman at (99, 100): peas = [(102, 100), []]

Pacman starts dropping peas (assuming it chooses to eat 2 foods at (102, 100) and (103, 100) first:

Pacman at (100, 100): peas = [[(102, 100), [(99, 100)]],
[(100, 97), []],
[(101, 102), []],
[(103, 100), []]]

…

Pacman at (102, 100): peas = [[(102, 100), [(99, 100), (100, 100), (101, 100)]],
[(100, 97), [(100, 100), (101, 100)]],
[(101, 102), [(100, 100), (101, 100)]],

[(103, 100), [(100, 100), (101, 100)]]]

Pacman at (103, 100): peas = [[(100, 97), [(100, 100), (101, 100), (102, 100)]],
[(101, 102), [(100, 100), (101, 100), (102, 100)]],
[(103, 100), [(100, 100), (101, 100), (102, 100)]]]

At (103, 100), we assume Pacman don't see any more food. Then it starts backtracking:

Pacman at (102, 100): peas = [[(100, 97), [(100, 100), (101, 100)]],
[(101, 102), [(100, 100), (101, 100)]]]

…

Pacman at (100, 100): peas = [[(100, 97), []],
[(101, 102), []]]

After finishing backtracking, Pacman continues to drop peas (assuming it chooses to eat the food at (100, 97) next):

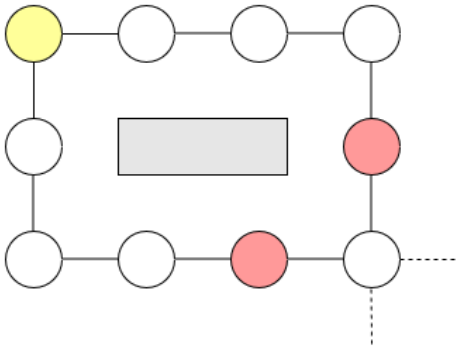
Pacman at (100, 99): peas = [[(100, 97), [(100, 100)]],
[(101, 102), [(100, 100)]]]

…

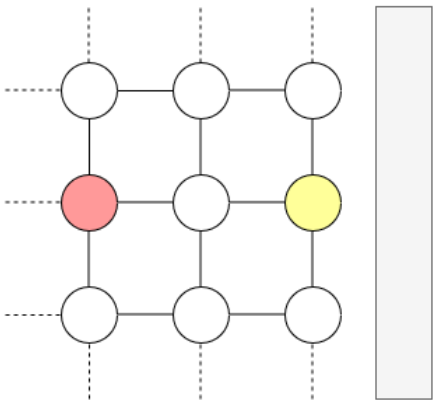
And so on.

c. Pacman losing cases:

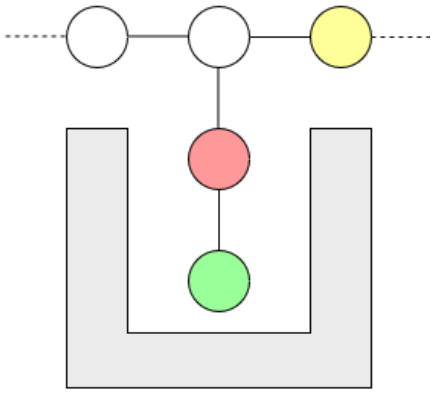
– Level 4: When Pacman at the corner and 2 monsters chase him. He's absolutely unable to escape.



– Level 4: When Pacman is pressed into the wall. At the same time, Monster and Pacman are parallel in around 3 cells. Then Pacman and Monster moving parallelly non-stop.



– Level 2: There is one food and one Monster on the map but the monster blocks the way which Pacman need to pass to be able to get food.



d. Bonus level 5:

We add level 5 with monster move randomly and Pacman is blind in order to see how intelligent Pacman can get all food and evade Monster on the way. Result is very good that Pacman can win the game.

5. References:

https://www.youtube.com/watch?v=a_TycD1RQxY&t=140s

https://www.youtube.com/watch?v=3sLV0OJLdns&t=36s

https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/