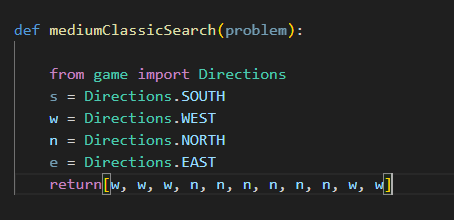
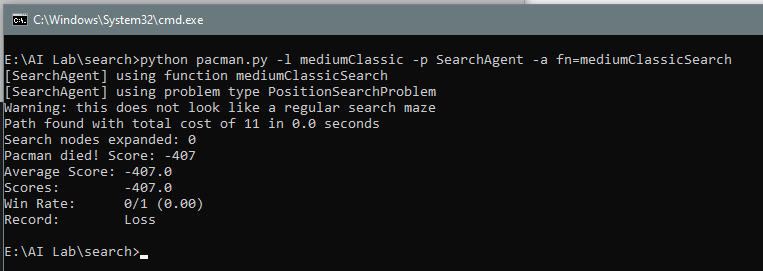
**Lab-03**

**Task-01**

Add code in search.py

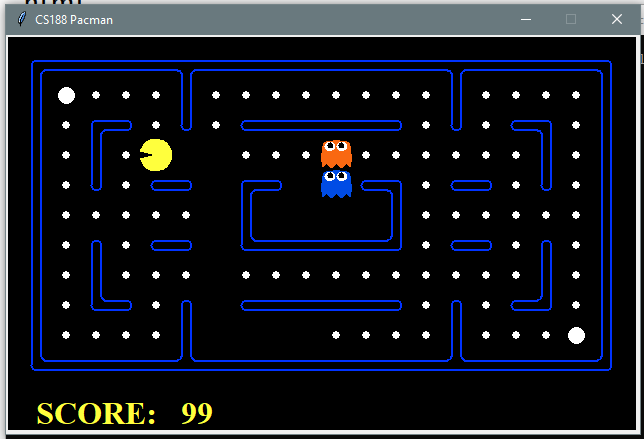


Write the following command in CMD



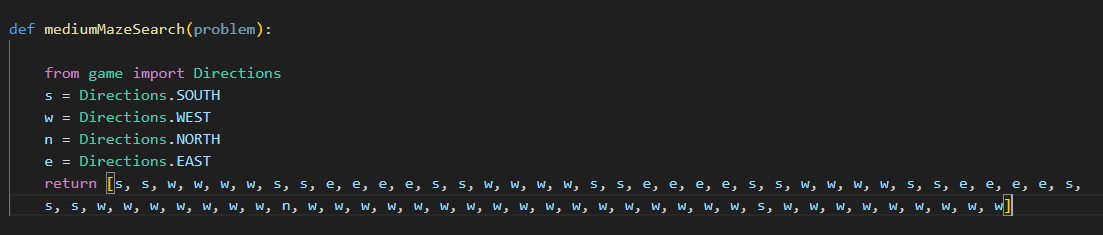
**Output**

Expected: Pac-man should eat at least one of his food successfully.

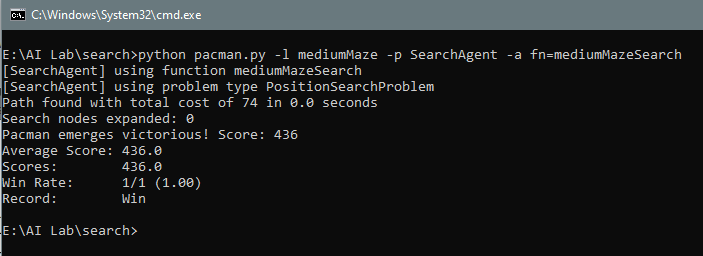


**Task-02**

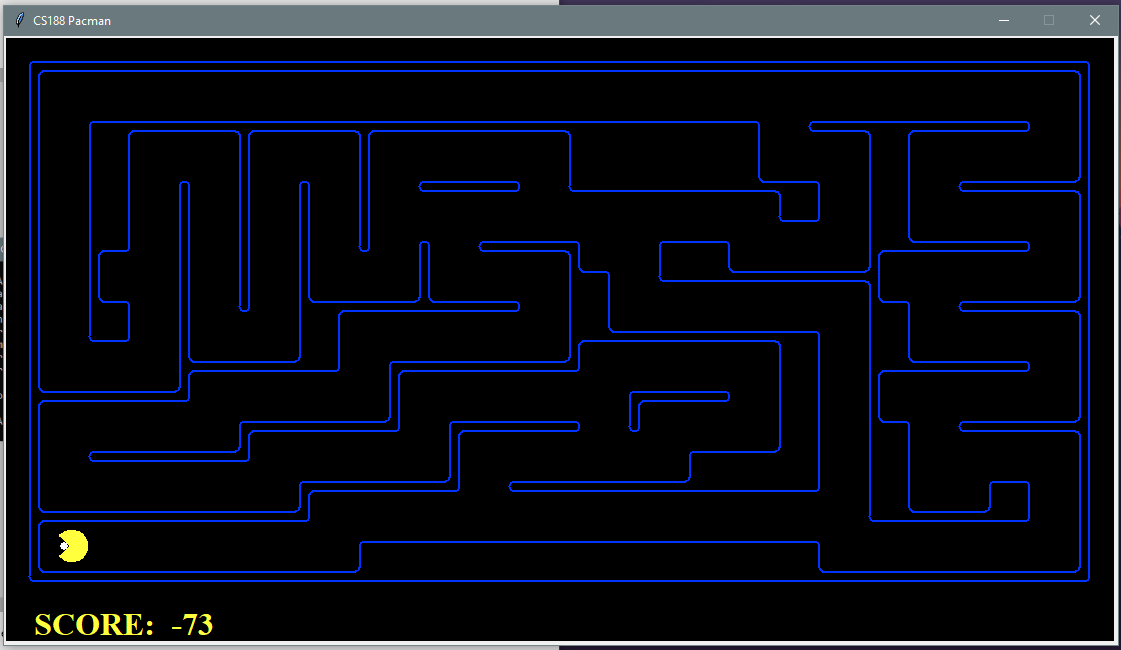
Add code in search.py



Write the following command in CMD

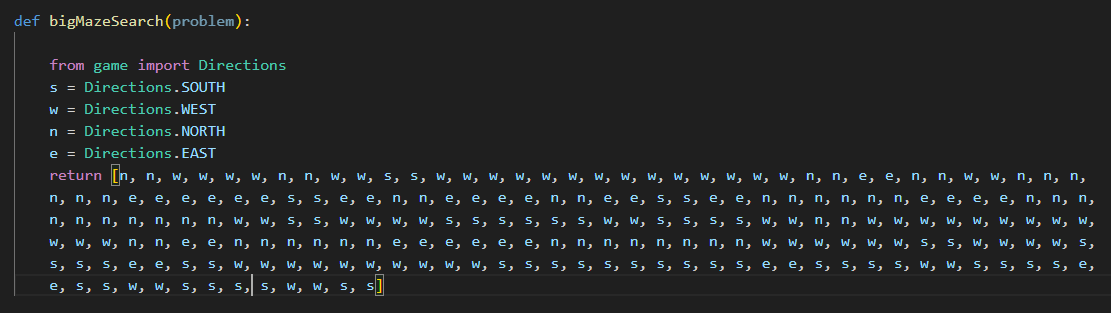


**Output**

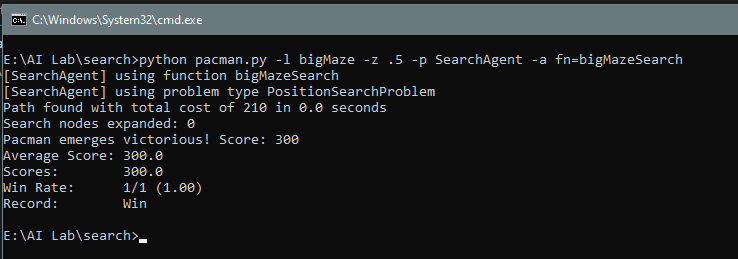
****

**Task-03**

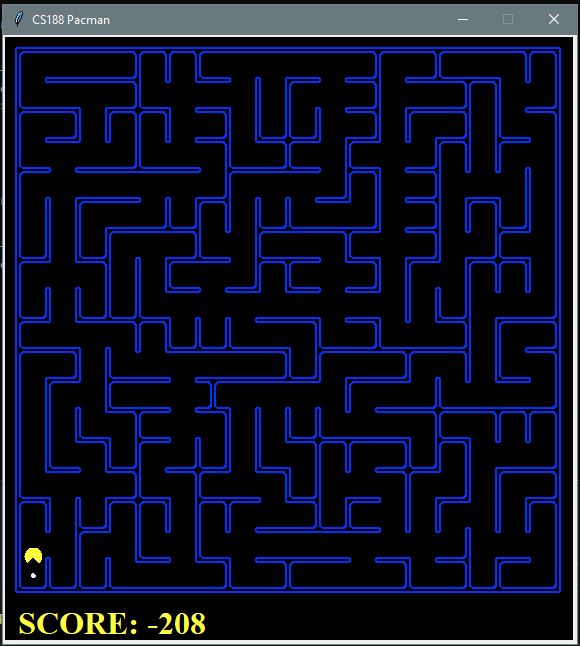
Add code in search.py



Write the following command in CMD



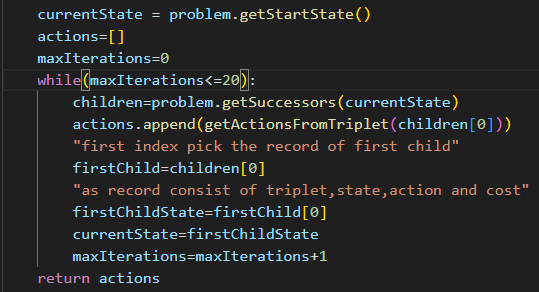
**Output**



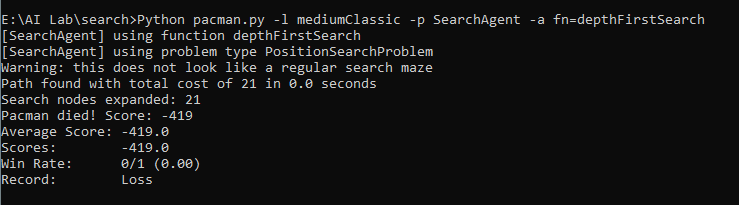
**Lab-04**

**Tak-01**

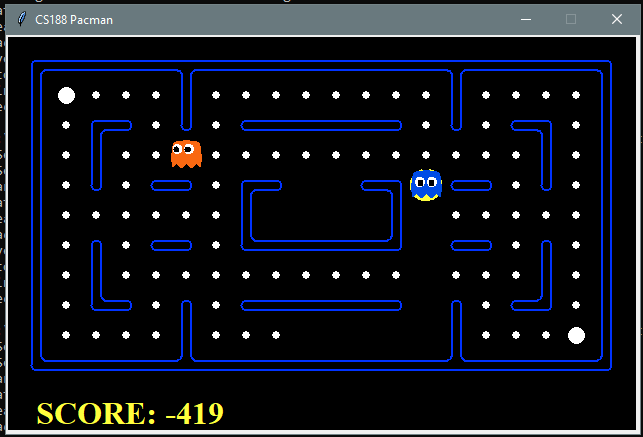
Max-Iteration is 20, 30 and 40.



Type the following command in CMD.



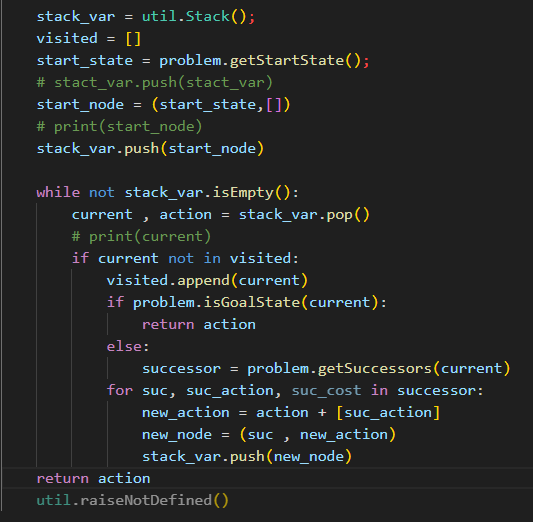
**Output**



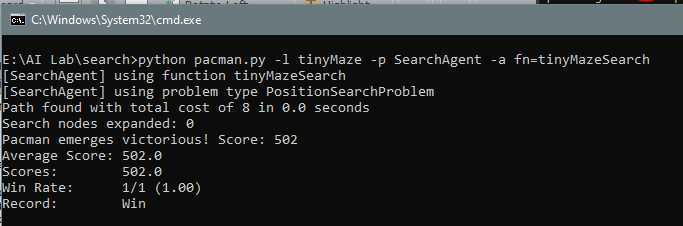
The Pac-man got stuck in loop because of **infinite** search space **possibilities**.

**Task-02**

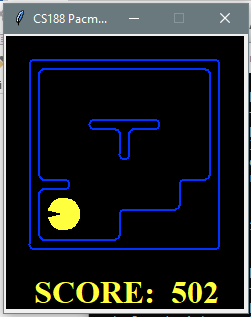
Add the following code in search.py



Type the following command in CMD

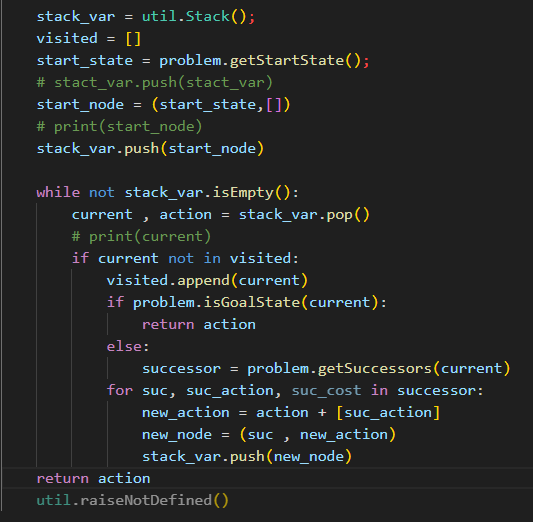


**Output**

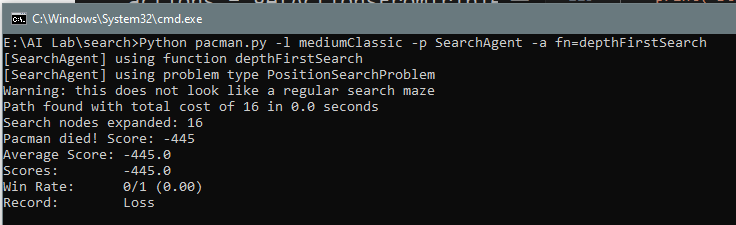


**Task-03**

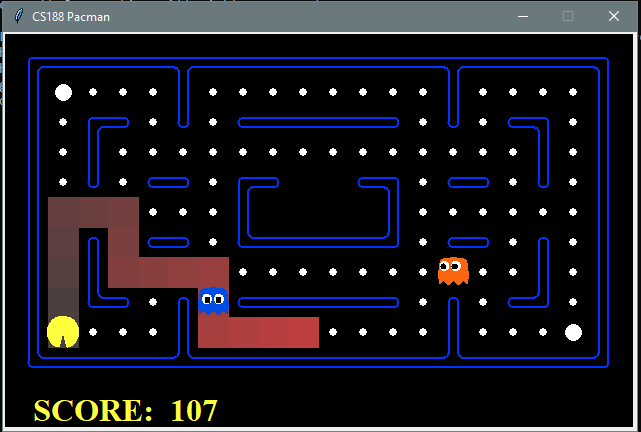
Adding the code in search.py



Type the following command in CMD.

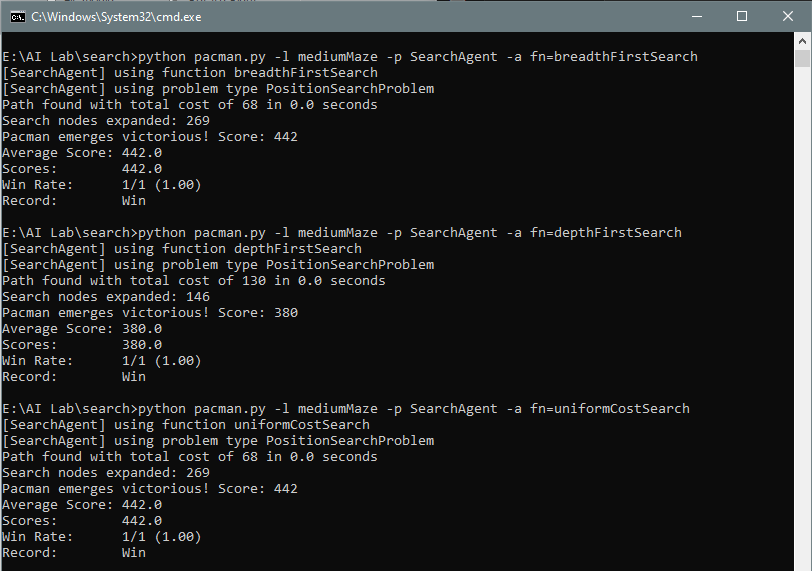


**Output**



**Lab-05**

**Task-01**

****

Fill the following table with the information.

|  |  |  |  |
| --- | --- | --- | --- |
|  | DFS | BFS | UCS |
| Total Cost | 130 | 68 | 68 |
| Nodes Expanded | 146 | 269 | 269 |
| Score | 380 | 442 | 442 |

▪ Why the total cost is Greater in DFS and Less in BFS?

BFS maintains a priority queue of the entire level while DFS just maintains a few pointers at each level by using a simple stack.

* Why Nodes Expanded are Greater in BFS and Less in DFS?

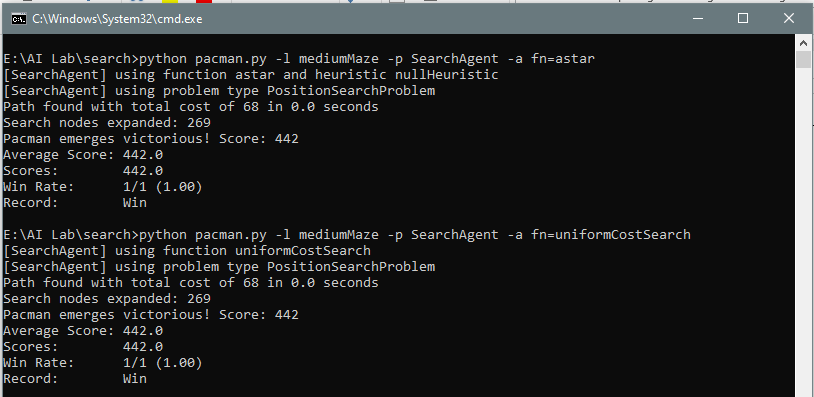
BFS stores all the nodes level by level (till they are processed). For an average graph, this might mean requiring larger space compared to DFS. BFS uses a larger amount of memory because it expands all children of a vertex and keeps them in memory.

▪ Write a note, which compares both algorithms in terms of these parameters.

* BFS is going to use more memory depending on the branching factor.  
  The space complexity of BFS is O ((branching factor) ^ (distance from source to destination)) as we have seen above.  
  If a graph is super wide and the goal state is not guaranteed to be closer to the start node then we need to think twice before performing a BFS, since that might not give an optimal result.
* BFS is better when the target is closer to the Source (like, looking for first degree and second-degree connections in the Facebook Friends network, or LinkedIn Connection Network).
* DFS is better when the target is far from the source (example: DFS is suitable for decision trees used in puzzle games).
* If you know a solution is not far from the start node, BFS would be better in most cases.  
  If solutions are located deep in the tree or graph, BFS could be impractical.

**Lab-06**

**Task-01**

****

Fill the following table with the information.

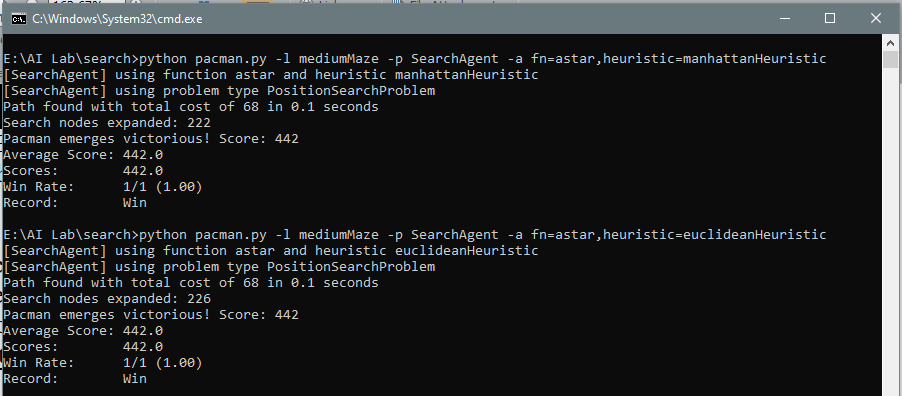
|  |  |  |
| --- | --- | --- |
|  | A\* | UCS |
| Total Cost | 68 | 68 |
| Nodes Expanded | 269 | 269 |
| Score | 442 | 442 |

UCS uses the evaluation function f(n) = g(n), where g(n) is the length of the path from the starting node to n, whereas A\* uses the evaluation function f(n)=g(n)+h(n), where g(n) means the same thing as in UCS and h(n), called the "heuristic" function, is an estimate of the distance from n to the goal node.

**Task-02**

UCS, Manhattan and Euclidean distance one by one and report your results.

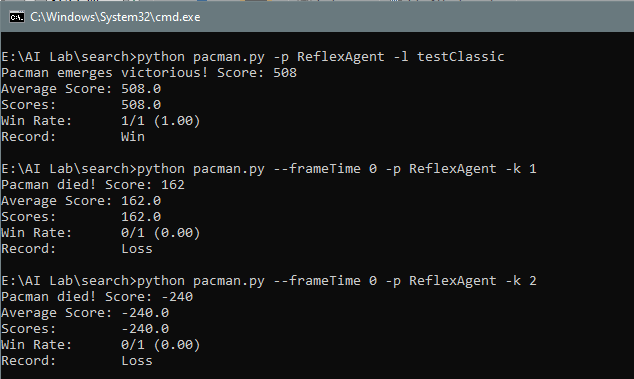
|  |  |  |  |
| --- | --- | --- | --- |
|  | Euclidean | Manhattan | UCS |
| Total Cost | 68 | 68 | 68 |
| Nodes Expanded | 226 | 222 | 269 |
| Score | 442 | 442 | 442 |

****

**Lab-07**

**Task- 01**

Improve the ReflexAgent in multiAgents.py to play respectably.

****

**Task- 02**

You will write an adversarial search agent in the provided MinimaxAgent class stub in multiAgents.py.

