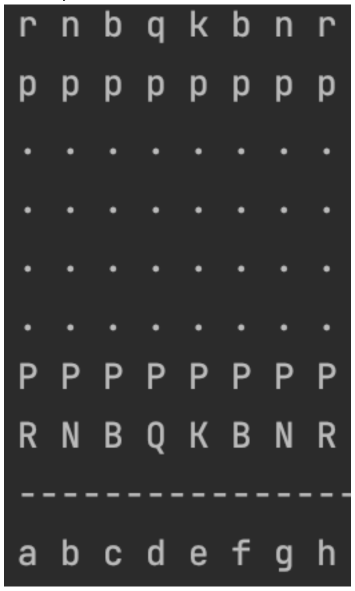
Chess

The simple chess game template was taken from **Dartmouth CS 76/176, Winter 201**9 course, **MiniMax** and **AlphaBeta** algorithms were added in addition to small modifications to the original code.

The game uses **chess** module in python, and illustrated in terminal as following text:



```
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R
----------------
a b c d e f g h
```

P = Paw, R = Rook, B = Bishop, N = Knight, Q = Queen, K = King

Lowercase and uppercase characters represent different colors/sides in the game.

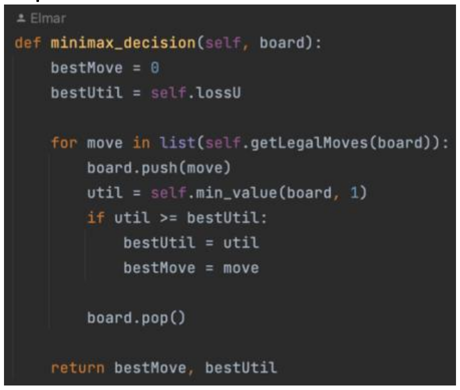
Minimax

The class requires 2 parameters for initialization: maximum depth to be explored and side (black/white).

The fundamental goal of minimax is to search the search tree for the best movements. If both players are rational and always maximize their own utility, the computer will explore ***depth*** moves away from the present location given a maximum depth of ***maxDepth***.

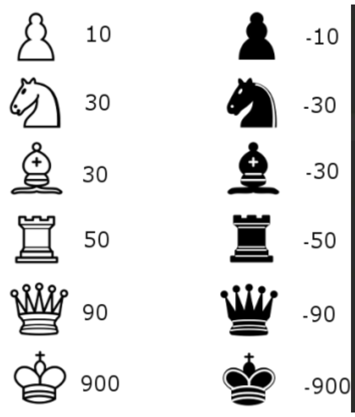
Minimax_decision() function iterates over all possible current legal moves on the board, and by using the .***push()*** method, it makes the move and then calculates utility on the move. If it’s better that the best utility value, appropriate updates are made to the values, and then .***pop()*** function is used to reverse the move.

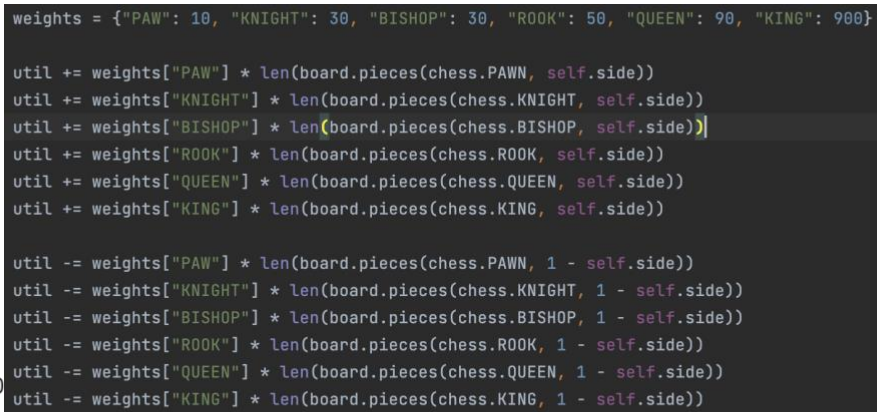
The cutoff function is implemented as we don't practically have enough computation power to explore all possible moves, and it makes sure only maximum depth of given ***maxDepth*** is explored.



```python
def minimax_decision(self, board):
    bestMove = 0
    bestUtil = self.lossU

    for move in list(self.getLegalMoves(board)):
        board.push(move)
        util = self.min_value(board, 1)
        if util >= bestUtil:
            bestUtil = util
            bestMove = move

        board.pop()

    return bestMove, bestUtil
```

Min_value() function will call **max_value()** on possible legal moves, and **max_value()** will keep track of the maximum of current highest utility and the utility calculated by **min_value()** function, and similarly, **min_value()** function will keep track of the minimum of current min utility value and the utility value calculated by **max_value()** function until depth limit is reached. This is checked by the ***cutoff()*** function, as we cannot practically compute all possible branches. In this demo program, the depth limit is set to 3. If the depth limit is reached, **calculate_util()** function is called to make an evaluation, and to do that, weights had to be assigned to figures:





```python
weights = {"PAW": 10, "KNIGHT": 30, "BISHOP": 30, "ROOK": 50, "QUEEN": 90, "KING": 900}

util += weights["PAW"] * len(board.pieces(chess.PAWN, self.side))
util += weights["KNIGHT"] * len(board.pieces(chess.KNIGHT, self.side))
util += weights["BISHOP"] * len(board.pieces(chess.BISHOP, self.side))
util += weights["ROOK"] * len(board.pieces(chess.ROOK, self.side))
util += weights["QUEEN"] * len(board.pieces(chess.QUEEN, self.side))
util += weights["KING"] * len(board.pieces(chess.KING, self.side))

util -= weights["PAW"] * len(board.pieces(chess.PAWN, 1 - self.side))
util -= weights["KNIGHT"] * len(board.pieces(chess.KNIGHT, 1 - self.side))
util -= weights["BISHOP"] * len(board.pieces(chess.BISHOP, 1 - self.side))
util -= weights["ROOK"] * len(board.pieces(chess.ROOK, 1 - self.side))
util -= weights["QUEEN"] * len(board.pieces(chess.QUEEN, 1 - self.side))
util -= weights["KING"] * len(board.pieces(chess.KING, 1 - self.side))
```

The weights multiplied by each piece number on the same side are added to the utility value, and then the ones on the opponent side are subtracted to get the final value.

AlphaBeta pruning

Most of the code base is similar to Minimax algorithm, but it comes with slight modifications to increase the overall performance. Alpha – lower bound, and Beta – higher bound is added to the code to ignore trees that are unlikely to have an optimal move (basically "pruning")

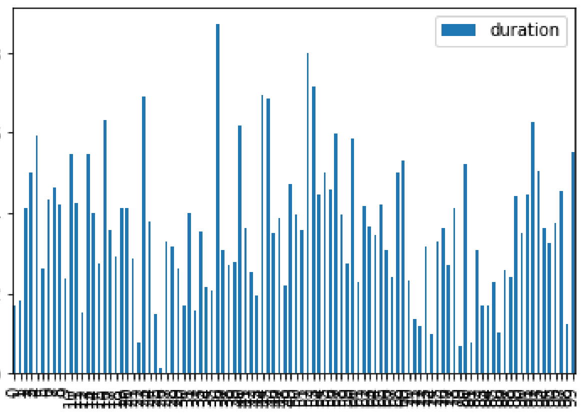
Elmar Karimov SE4GD '24 / AI 2022 Fall HW1

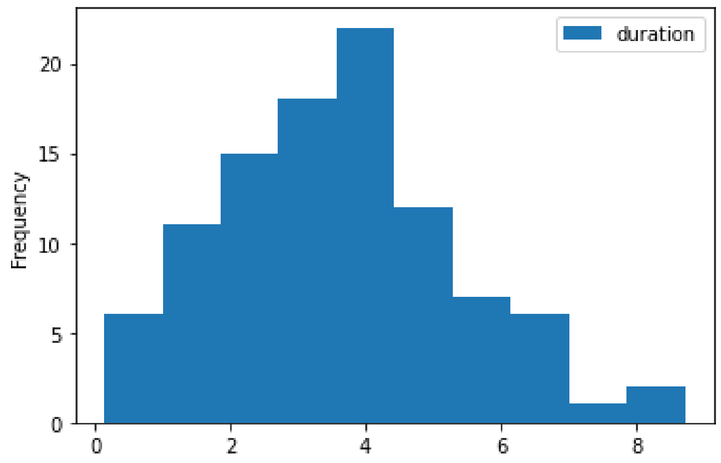
Extra:

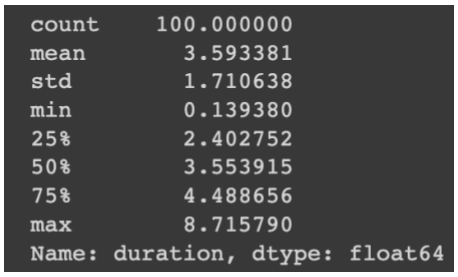
When choosing the first moves, which are also called "openings" some pre-defined moves could be made. They are commented under ***choose_move()*** function in **AlphaBetaAI.py** file, which also uses the pre-defined moves in **constants.py** file.

Below the side by side comparison of minimax and alphabeta algorithms based on 100 played chess games against a random agent. In summary, minimax takes 13.7 seconds while alphabeta takes 3.6 seconds to win the game on average.

Minimax

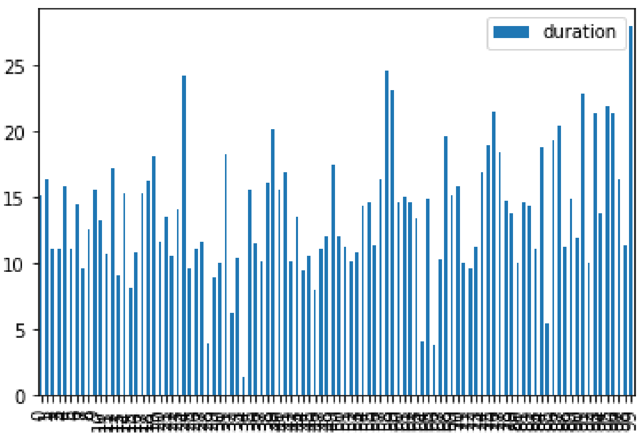


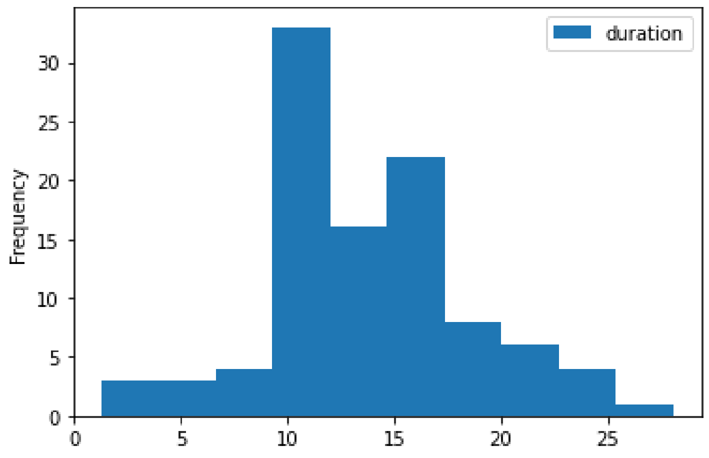


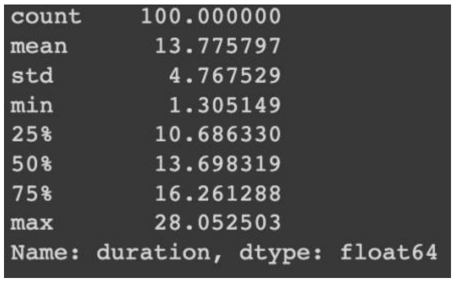


```
count    100.000000
mean       3.593381
std        1.710638
min        0.139380
25%        2.402752
50%        3.553915
75%        4.488656
max        8.715790
Name: duration, dtype: float64
```

AlphaBeta





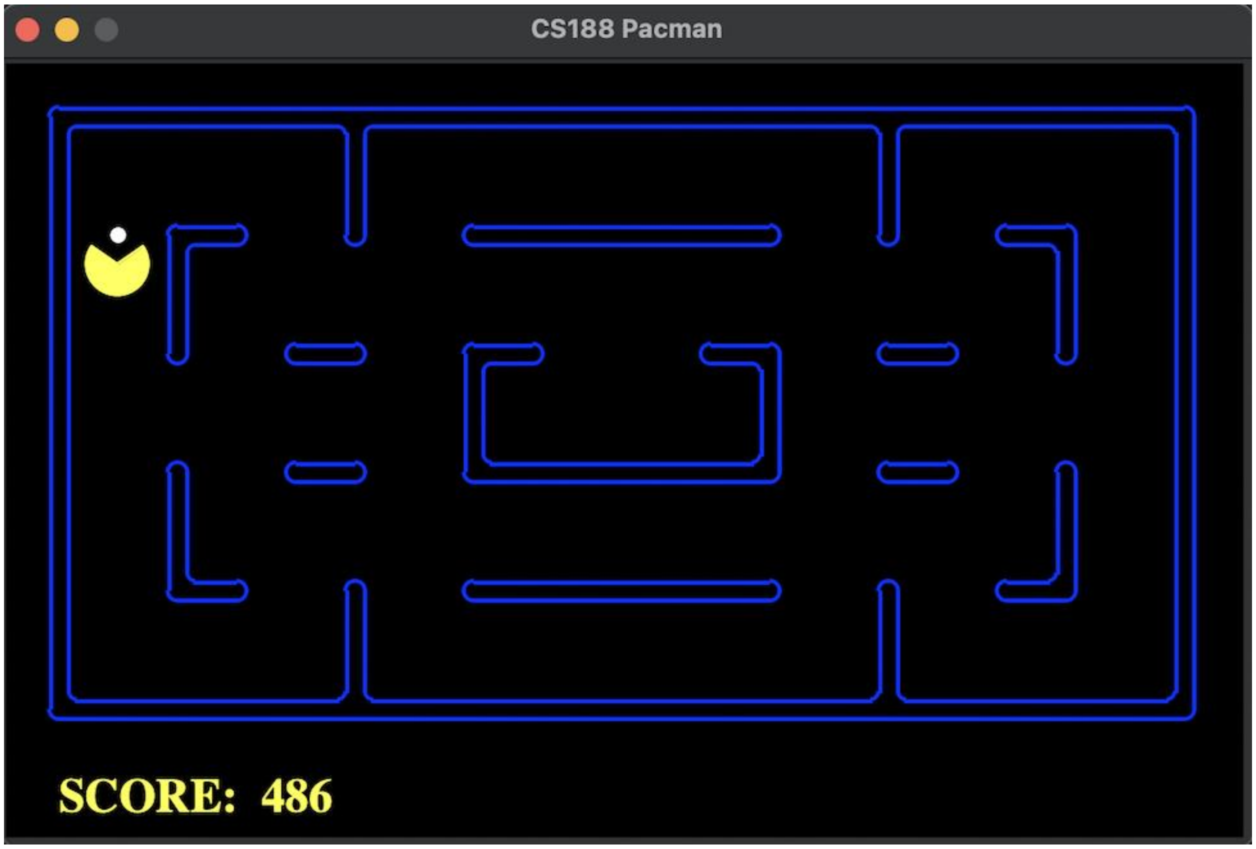


```
count    100.000000
mean      13.775797
std        4.767529
min        1.305149
25%       10.686330
50%       13.698319
75%       16.261288
max       28.052503
Name: duration, dtype: float64
```

References:

https://pypi.org/project/chess/
https://www.cs.dartmouth.edu/~devin/cs76/03_chess/chess.html
https://en.wikipedia.org/wiki/Chess_opening
https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/
https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
Russell, S. J., & Norvig, P. (1995). Artificial intelligence: a modern approach. Englewood Cliffs, N.J., Prentice Hall.

Pacman



The project was taken from http://ai.berkeley.edu/project_overview.html, BFS and rest of the A* algorithm is added to the code.

BFSAgent

When the algorithm is initialized, reached is set false indicating the algorithm has not yet completed the search. It also has a ***solution*** stack to save the path to solution.

getAction() is the main function of this agent, and it first checks whether the agent has reached destination, in our game, destination is the food.
gameState provides a number of functions, including getLegalPacmanActions() and after getting all those possible actions, their states are stored in a queue to be explored by the BFS algorithm.

Inside the while loop, a state waiting to be explored is popped from the queue, and added to the visited states list (defined by its position). If the current state is a winning state, it is pushed to the solution stack. Otherwise, BFS continues exploring the remaining states in the queue, including their successors (this is the way BFS expands). When the solution is found, reached flag is enabled.

Astar

Algorithm is started by pushing the start node to the priority queue along with its cost, calculated by **manhattandistance**. While the priorityqueue is not empty, an element is popped from the queue and added to the visited list (again, defined by its position). If it is on the route to the goal node, it is pushed to the solution path and the pointer goes back to the parent until it reaches the start of the path. Either way, all nodes on the map are explored, and total cost is calculated and added to the state, and the state is pushed to the queue to continue exploration.

To run the algorithms, command line arguments were used:

python pacman.py -p AstarAgent
python pacman.py -p BFSAgent

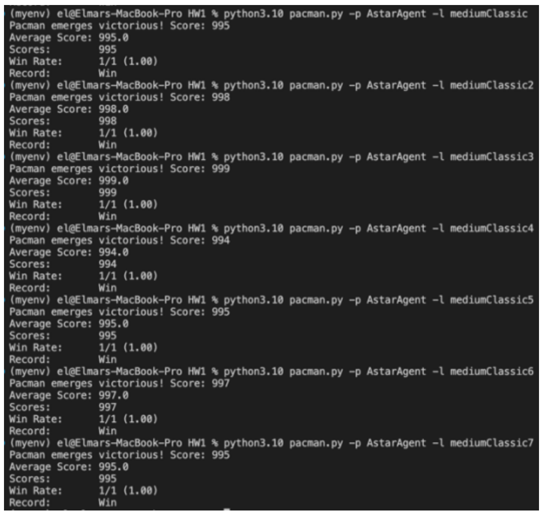
note: on my machine it ran only with **python3.10**

I noticed that it is possible to use custom maps with -l flag, therefore I placed the food in different locations on the same map and created 7 variations:

python pacman.py -p AstarAgent -l mediumClass
python pacman.py -p BFSAgent -l mediumClass

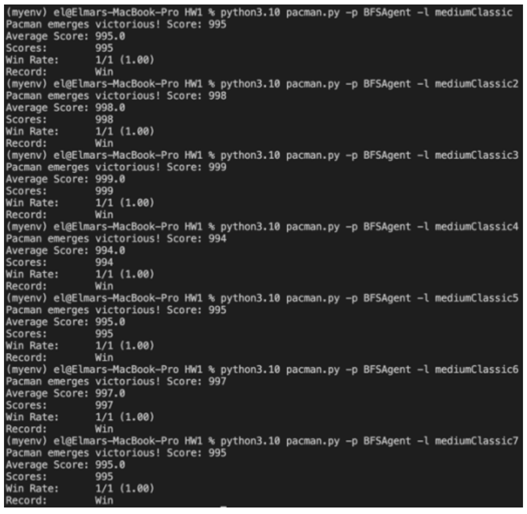
Each run gives scores obtained by the algorithm, below is the statistics of the scores:

AstarAgent



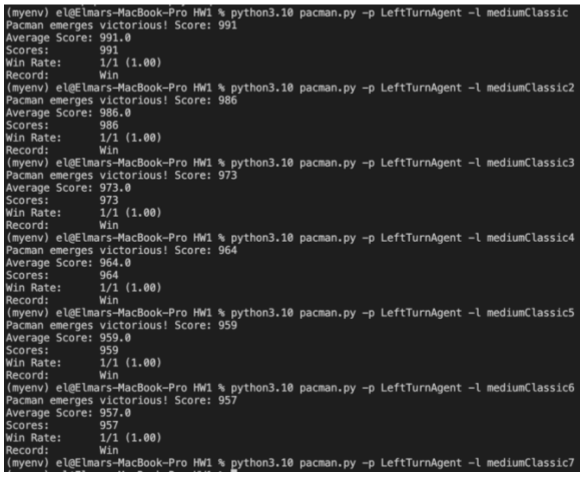
```
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p AstarAgent -l mediumClassic
Pacman emerges victorious! Score: 995
Average Score: 995.0
Scores:        995
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p AstarAgent -l mediumClassic2
Pacman emerges victorious! Score: 998
Average Score: 998.0
Scores:        998
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p AstarAgent -l mediumClassic3
Pacman emerges victorious! Score: 999
Average Score: 999.0
Scores:        999
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p AstarAgent -l mediumClassic4
Pacman emerges victorious! Score: 994
Average Score: 994.0
Scores:        994
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p AstarAgent -l mediumClassic5
Pacman emerges victorious! Score: 995
Average Score: 995.0
Scores:        995
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p AstarAgent -l mediumClassic6
Pacman emerges victorious! Score: 997
Average Score: 997.0
Scores:        997
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p AstarAgent -l mediumClassic7
Pacman emerges victorious! Score: 995
Average Score: 995.0
Scores:        995
Win Rate:      1/1 (1.00)
Record:        Win
```

BFSAgent



```
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p BFSAgent -l mediumClassic
Pacman emerges victorious! Score: 995
Average Score: 995.0
Scores:        995
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p BFSAgent -l mediumClassic2
Pacman emerges victorious! Score: 998
Average Score: 998.0
Scores:        998
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p BFSAgent -l mediumClassic3
Pacman emerges victorious! Score: 999
Average Score: 999.0
Scores:        999
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p BFSAgent -l mediumClassic4
Pacman emerges victorious! Score: 994
Average Score: 994.0
Scores:        994
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p BFSAgent -l mediumClassic5
Pacman emerges victorious! Score: 995
Average Score: 995.0
Scores:        995
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p BFSAgent -l mediumClassic6
Pacman emerges victorious! Score: 997
Average Score: 997.0
Scores:        997
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p BFSAgent -l mediumClassic7
Pacman emerges victorious! Score: 995
Average Score: 995.0
Scores:        995
Win Rate:      1/1 (1.00)
Record:        Win
```

I also tried the pre-implemented LeftStarAgent, which only turns left:



```
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p LeftTurnAgent -l mediumClassic
Pacman emerges victorious! Score: 991
Average Score: 991.0
Scores:        991
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p LeftTurnAgent -l mediumClassic2
Pacman emerges victorious! Score: 986
Average Score: 986.0
Scores:        986
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p LeftTurnAgent -l mediumClassic3
Pacman emerges victorious! Score: 973
Average Score: 973.0
Scores:        973
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p LeftTurnAgent -l mediumClassic4
Pacman emerges victorious! Score: 964
Average Score: 964.0
Scores:        964
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p LeftTurnAgent -l mediumClassic5
Pacman emerges victorious! Score: 959
Average Score: 959.0
Scores:        959
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p LeftTurnAgent -l mediumClassic6
Pacman emerges victorious! Score: 957
Average Score: 957.0
Scores:        957
Win Rate:      1/1 (1.00)
Record:        Win
(myenv) el@Elmars-MacBook-Pro HW1 % python3.10 pacman.py -p LeftTurnAgent -l mediumClassic7
```

References

http://ai.berkeley.edu/project_overview.html
https://en.wikipedia.org/wiki/Pac-Man
https://en.wikipedia.org/wiki/Breadth-first_search