<u>BONUS QUESTION LAB 3B</u>
<u>Group - 21</u>

Gautam Kumar Mahar
Roll. N. 2103114
Kanwar Raj Singh
Roll. N. 1903122

Optional Questions (for bonus credits):

Question 7: In Question 6, you built an approximate Q-learning agent using the features provided by the SimpleExtractor defined in featureExtractors.py. Take some time to understand the SimpleExtractor class containing the getFeatures function. There is scope for improvement of these features as they do not consider the scaredTimers/larger food pellets. Your task is to improve the Approximate Q-learning agent for Pacman by building your own advanced extractor class/features in featureExtractors.py (you are free to construct any features you like). For this you will need to create a new class for the implementation (similar to how the SimpleExtractor class is written in featureExtractors.py). See if you can demonstrate cases where your features are clearly making a difference. Additional file to be submitted for this question: featureExtractors.py

<u>**OUTPUT**</u>

```
┌──(gautamop㉿gautamop)-[~/…/CS330/LAB_ASSIGNMENT_3B/BONUS/reinforcement]
└─$ python pacman.py -p ApproximateQAgent -a extractor=AdvancedExtractor -x 50 -n 60 -l mediumClassic -q
Beginning 50 episodes of Training
Training Done (turning off epsilon and alpha)
---------------------------------------------
{'closest-food': -1.7105041097018092, '#-of-ghosts-1-step-away': -154.58316026973, 'food-in-North': 229.6835573609613, 'food-in-South': 104.46342650678011, 'food-in-East': 133.64906
00335735, 'food-in-West': 121.75712133005696, 'scared-ghost-nearby': 209.51863268913164}
Pacman emerges victorious! Score: 1733
Pacman emerges victorious! Score: 1533
Pacman emerges victorious! Score: 1733
Pacman emerges victorious! Score: 1527
Pacman emerges victorious! Score: 1310
Pacman emerges victorious! Score: 1492
Pacman emerges victorious! Score: 1523
Pacman emerges victorious! Score: 1330
Pacman emerges victorious! Score: 1941
Pacman emerges victorious! Score: 1333
Average Score: 1545.5
Scores:        1733.0, 1533.0, 1733.0, 1527.0, 1310.0, 1492.0, 1523.0, 1330.0, 1941.0, 1333.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

For running this file

Unset
```
python pacman.py -p ApproximateQAgent -a extractor=AdvancedExtractor
 -x 50 -n 60 -l mediumGrid

python pacman.py -p ApproximateQAgent -a extractor=AdvancedExtractor
-x 50 -n 60 -l mediumClassic
```

Solution In featureExtractors.py

```python
class AdvancedExtractor(FeatureExtractor):
    """
    Returns advanced features for a Pacman agent:
    - Distance to the nearest food
    - Whether a ghost is one step away
    - Whether food is available in a given direction
    - Whether a scared ghost is nearby
    """

    def getFeatures(self, state, action):
        # Extract relevant information from the game state
        food = state.getFood()
        walls = state.getWalls()
        ghosts = state.getGhostPositions()
        pacman_position = state.getPacmanPosition()
        ghost_states = state.getGhostStates()

        features = util.Counter()

        # Compute the location of pacman after taking the action
        x, y = pacman_position
        dx, dy = Actions.directionToVector(action)
        next_x, next_y = int(x + dx), int(y + dy)

        # Feature: Distance to the nearest food
        dist = closestFood((next_x, next_y), food, walls)
        if dist is not None:
            features["closest-food"] = float(dist) / (walls.width * walls.height)

        # Feature: Whether a ghost is one step away
        features["#-of-ghosts-1-step-away"] = sum(
            (next_x, next_y) in Actions.getLegalNeighbors(g.getPosition(), walls)
            for g in ghost_states if not g.scaredTimer
        )

        # Feature: Whether food is available in each direction
        for direction in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
            dx, dy = Actions.directionToVector(direction)
            next_x, next_y = int(x + dx), int(y + dy)
            features[f"food-in-{direction}"] = int(food[next_x][next_y])

        # Feature: Whether a scared ghost is nearby
        for ghost_state in ghost_states:
            if ghost_state.scaredTimer > 0:
                ghost_position = ghost_state.getPosition()
                ghost_distance = util.manhattanDistance(pacman_position, ghost_position)
                if ghost_distance <= 5:
                    features["scared-ghost-nearby"] = 1.0
                    break

        features.divideAll(10.0)
        return features
```

In this advanced feature extractor, we have added features to consider:

- Extract relevant information from the game state, such as food locations, wall locations, ghost positions, Pacman's position, and ghost states.

- Calculate the expected position of Pacman after taking the given action.

- Calculate the distance to the nearest food and scale it to a value between 0 and 1 using the width and height of the maze.

- Check if a ghost is one step away from Pacman, considering only non-scared ghosts.

- Check if food is available in each of the four cardinal directions (north, south, east, west) relative to Pacman's current position.

- Determine if a scared ghost is nearby by calculating the Manhattan distance between Pacman and each ghost with a positively scared timer. If a scared ghost is within a distance of 5, set the "scared-ghost-nearby" feature to 1.0.

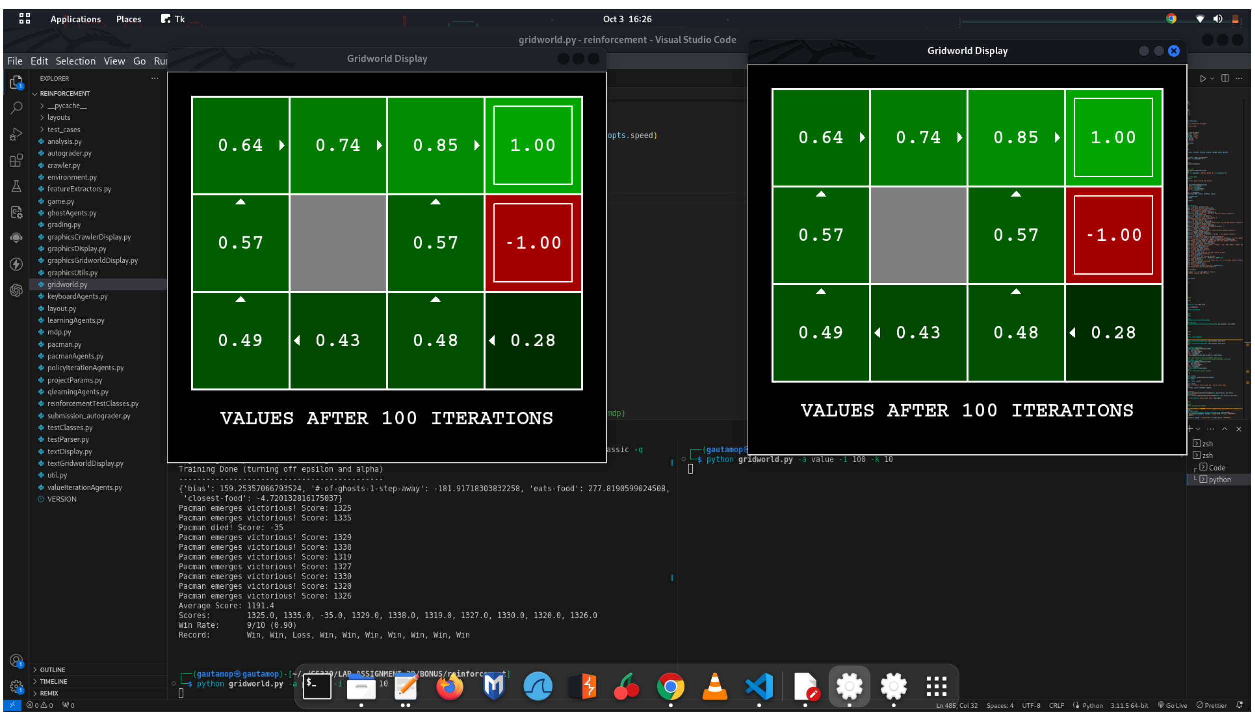
- Divide all the feature values by 10.0 to normalize them to a range between 0 and 1.

Question 8: In Question 1, you developed an agent that does Value Iteration. Now develop an agent that does Policy Iteration. Compare the speed of policy iteration vs. value iteration on Gridworld. The implementation details are up to you - it could be in a new file or in existing files. When you make a submission please provide documentation to locate your code.

SOLUTION - Code Included in <u>policyIterationAgents.py</u>

- For Running Purposes Use My <u>gridworld.py</u> file.

<u>OUTPUT</u>



<u>Left Side</u>

<u>policyIterationAgents.py</u>

python gridworld.py -a policy -i 100 -k 10

<u>Right Side</u>

<u>valueIterationAgents.py</u>

python gridworld.py -a value -i 100 -k 10

Components and methods used in the PolicyIterationAgent file:

<u>MDP and ValueEstimationAgent:</u>

The PolicyIterationAgent inherits from ValueEstimationAgent, indicating that it's an agent for estimating value functions.

<u>Initialization:</u>
The agent is initialized with parameters such as the MDP, discount factor, and the number of iterations for policy evaluation and improvement.
Policy Initialization:

The agent initializes the policy arbitrarily. For non-terminal states, it assigns the first possible action from getPossibleActions(state) as the initial policy.

Policy Evaluation:

The runPolicyEvaluation method is used to evaluate the current policy and update the state values in self.policyValues. It uses linear algebra techniques to solve the Bellman equation.

Policy Improvement:

The runPolicyImprovement method is used to improve the policy based on the current state values in self. policy values. It updates the policy by selecting the action with the highest Q-value for each state.

Q-Value Computation:

The computeQValueFromValues method computes the Q-value of taking a specific action in a given state.

getValue, getQValue, getPolicy, getAction:

These methods are provided to interact with the agent and query its values, Q-values, policy, and selected actions for specific states.

THANK YOU