COMP6065 – Artificial Intelligence (L5AC)

Final Project Report

Thedy Dinata - 1901531054

Odd Semester 2017/2018

Thursday, 18th January 2018

Table of Contents

I. Introduction

Since long ago, the interests of games bots have gained lot of attention, especially after Google DeepMind, Alphabet Inc. created AlphaGo, a robot that are capable of learning and even beating Pro Go player.

This project is a basic learning of what machine learning are capable of for games, for this project I specially choose Pacman. This program is called Automated Pacman. Based on learning, the pacman will learn and train of best possible move for him to get the win.

This project is a project that I based on tutorial from Berkeley UC.

II. Solution Features

Automated Pacman uses Reinforcement Learning or QLearning. The model is made by using Tkinter, python's de-facto standard GUI, and python version 2.x(tested on python 2.7).

III. Solution Design Architecture

To build this Automated Pacman, I am using Python programming language version 2.7 and Tkinter(python packages). For the machine learning I'm using QLearning, a model-free reinforcement learning technique that can be used to find an optimal action-selection policy for Markov Decision Process.

The specification of the hardware used to train this program is :
CPU: Intel® Core™ i7-4720HQ CPU @ 2.60Ghz 2.59 GHZ
RAM: 12.0 GB
GPU: NVIDIA GeForce GTX 950M

IV. Program Manual

A. Code Snippets

Train function

```python
class QLearningAgent(ReinforcementAgent):
  """
    Q-Learning Agent

    Functions you should fill in:
      - getQValue
      - getAction
      - getValue
      - getPolicy
      - update

    Instance variables you have access to
      - self.epsilon (exploration prob)
      - self.alpha (learning rate)
      - self.discount (discount rate)

    Functions you should use
      - self.getLegalActions(state)
        which returns legal actions
        for a state
  """
  def __init__(self, **args):
    "You can initialize Q-values here..."
    ReinforcementAgent.__init__(self, **args)

    "*** YOUR CODE HERE ***"
    self.qValues = util.Counter()
    print "ALPHA", self.alpha
    print "DISCOUNT", self.discount
    print "EXPLORATION", self.epsilon

  def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we never seen
      a state or (state,action) tuple
    """
    "*** YOUR CODE HERE ***"
    return self.qValues[(state, action)]


  def getValue(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
```

```python
        """*** YOUR CODE HERE ***"""
        possibleStateQValues = util.Counter()
        for action in self.getLegalActions(state):
            possibleStateQValues[action] = self.getQValue(state, action)

        return possibleStateQValues[possibleStateQValues.argMax()]

    def getPolicy(self, state):
        """
          Compute the best action to take in a state.  Note that if there
          are no legal actions, which is the case at the terminal state,
          you should return None.
        """
        """*** YOUR CODE HERE ***"""
        possibleStateQValues = util.Counter()
        possibleActions = self.getLegalActions(state)
        if len(possibleActions) == 0:
            return None

        for action in possibleActions:
            possibleStateQValues[action] = self.getQValue(state, action)

        if possibleStateQValues.totalCount() == 0:
            return random.choice(possibleActions)
        else:
            return possibleStateQValues.argMax()

    def getAction(self, state):
        """
          Compute the action to take in the current state.  With
          probability self.epsilon, we should take a random action and
          take the best policy action otherwise.  Note that if there are
          no legal actions, which is the case at the terminal state, you
          should choose None as the action.

          HINT: You might want to use util.flipCoin(prob)
          HINT: To pick randomly from a list, use random.choice(list)
        """
        # Pick Action
        legalActions = self.getLegalActions(state)
        action = None
        """*** YOUR CODE HERE ***"""
        if len(legalActions) > 0:
            if util.flipCoin(self.epsilon):
                action = random.choice(legalActions)
            else:
```

```python
        action = self.getPolicy(state)

    return action

  def update(self, state, action, nextState, reward):
    """
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here

      NOTE: You should never call this function,
      it will be called on your behalf
    """
    "*** YOUR CODE HERE ***"
    print "State: ", state, " , Action: ", action, " , NextState: ", nextState, " , Reward: ", reward
    print "QVALUE", self.getQValue(state, action)
    print "VALUE", self.getValue(nextState)
    self.qValues[(state, action)] = self.getQValue(state, action) + self.alpha * (reward + self.discount * self.getValue(nextState) - self.getQValue(state, action))

class PacmanQAgent(QLearningAgent):
  "Exactly the same as QLearningAgent, but with different default parameters"

  def __init__(self, epsilon=0.05,gamma=0.8,alpha=0.2, numTraining=0, **args):
    """
    These default parameters can be changed from the pacman.py command line.
    For example, to change the exploration rate, try:
        python pacman.py -p PacmanQLearningAgent -a epsilon=0.1

    alpha    - learning rate
    epsilon  - exploration rate
    gamma    - discount factor
    numTraining - number of training episodes, i.e. no learning after these many episodes
    """
    args['epsilon'] = epsilon
    args['gamma'] = gamma
    args['alpha'] = alpha
    args['numTraining'] = numTraining
    self.index = 0  # This is always Pacman
    QLearningAgent.__init__(self, **args)

  def getAction(self, state):
    """
    Simply calls the getAction method of QLearningAgent and then
    informs parent of action for Pacman.  Do not change or remove this
    method.
```

```python
    action = QLearningAgent.getAction(self,state)
    self.doAction(state,action)
    return action
```

B. Training & Results

1. 5000 Training – 10 Games

```
Reinforcement Learning Status:
        Completed 4600 out of 5000 training episodes
        Average Rewards over all training: 1094.36
        Average Rewards for last 100 episodes: 1038.84
        Episode took 43.37 seconds
Reinforcement Learning Status:
        Completed 4700 out of 5000 training episodes
        Average Rewards over all training: 1093.07
        Average Rewards for last 100 episodes: 1033.50
        Episode took 43.45 seconds
Reinforcement Learning Status:
        Completed 4800 out of 5000 training episodes
        Average Rewards over all training: 1090.88
        Average Rewards for last 100 episodes: 988.05
        Episode took 45.06 seconds
Reinforcement Learning Status:
        Completed 4900 out of 5000 training episodes
        Average Rewards over all training: 1093.52
        Average Rewards for last 100 episodes: 1220.50
        Episode took 47.76 seconds
Reinforcement Learning Status:
        Completed 5000 out of 5000 training episodes
        Average Rewards over all training: 1091.85
        Average Rewards for last 100 episodes: 1009.59
        Episode took 43.05 seconds
Training Done (turning off epsilon and alpha)
---------------------------------------------
Pacman emerges victorious! Score: 1924
Pacman emerges victorious! Score: 1543
Pacman emerges victorious! Score: 1524
Pacman emerges victorious! Score: 1710
Pacman emerges victorious! Score: 1528
Pacman emerges victorious! Score: 1731
Pacman emerges victorious! Score: 1929
Pacman emerges victorious! Score: 1531
Pacman emerges victorious! Score: 1323
Pacman emerges victorious! Score: 1729
Average Score: 1647.2
Scores:        1924, 1543, 1524, 1710, 1528, 1731, 1929, 1531, 1323, 1729
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

2. 5000 Training – 100 Games

```
Reinforcement Learning Status:
        Completed 100 test episodes
        Average Rewards over testing: 1348.02
        Average Rewards for last 100 episodes: 1348.02
        Episode took 3369.66 seconds
Average Score: 1348.02
Scores:        1738, 1524, 1318, 148, 1931, 1931, 1324, 1320, 1335, 1930, 484, 1526, 1717, 1731, 1723, 1515, 1325, 1931,
 1519, 1519, 1937, 1714, 1510, 1731, 1517, 1526, 83, 1328, 310, 1316, -43, 92, 1519, 1722, 1538, 1731, 1324, 1722, 1518,
 1731, 1706, 1303, 645, 1529, -25, 1531, 1725, 1331, 1340, 1727, 1331, 1931, 1733, 1536, 1531, 1530, 1513, 1531, 1530, 2
77, 1331, 1936, 1515, 1728, 421, 1513, 1544, 1322, 22, 1531, 294, 1328, 1331, 1538, 1531, 1301, 1726, 1732, 1331, 1336,
1742, 1311, 1539, 1525, -25, 1516, 1526, 1521, 1741, 1521, 83, 1731, 1545, 458, 1732, 1935, 1531, 118, 1345, 1526
Win Rate:      84/100 (0.84)
Record:        Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
, Win, Win, Win, Win, Win, Loss, Win, Loss, Win, Loss, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win
, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Loss, Win, Win, Loss, Win, Win,
Win, Loss, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Lo
ss, Win, Win, Loss, Win, Win, Win, Loss, Win, Win
PS G:\BINUS\Pacman> python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 5000 -n 5100 -l mediumClassic
```

3. 5000 Training – 1000 Games

```
Win Rate:      866/1000 (0.87)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win
, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Loss, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Loss
, Win, Loss, Win, Win, Win, Win, Loss, Win, Win, Loss, Win, Loss, Win, Win, Win, Win, Loss, Win, Win, Win, Loss, Win, Win, Win, Win, Win, W
in, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Wi
n, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Loss, Loss, Win, Win, Win, Win, Win, Win, Loss, Win, W
```

4. 10000 Training – 10 Games

```
Reinforcement Learning Status:
        Completed 9600 out of 10000 training episodes
        Average Rewards over all training: 1108.10
        Average Rewards for last 100 episodes: 1051.09
        Episode took 42.16 seconds
Reinforcement Learning Status:
        Completed 9700 out of 10000 training episodes
        Average Rewards over all training: 1106.60
        Average Rewards for last 100 episodes: 963.19
        Episode took 40.71 seconds
Reinforcement Learning Status:
        Completed 9800 out of 10000 training episodes
        Average Rewards over all training: 1106.79
        Average Rewards for last 100 episodes: 1124.78
        Episode took 46.56 seconds
Reinforcement Learning Status:
        Completed 9900 out of 10000 training episodes
        Average Rewards over all training: 1106.04
        Average Rewards for last 100 episodes: 1032.78
        Episode took 43.63 seconds
Reinforcement Learning Status:
        Completed 10000 out of 10000 training episodes
        Average Rewards over all training: 1105.50
        Average Rewards for last 100 episodes: 1052.09
        Episode took 42.53 seconds
Training Done (turning off epsilon and alpha)
---------------------------------------------
Pacman emerges victorious! Score: 1521
Pacman emerges victorious! Score: 1332
Pacman emerges victorious! Score: 1734
Pacman emerges victorious! Score: 1732
Pacman emerges victorious! Score: 1531
Pacman emerges victorious! Score: 1314
Pacman emerges victorious! Score: 1531
Pacman emerges victorious! Score: 1526
Pacman emerges victorious! Score: 1337
Pacman emerges victorious! Score: 1724
Average Score: 1528.2
Scores:        1521, 1332, 1734, 1732, 1531, 1314, 1531, 1526, 1337, 1724
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

5. 10000 Training – 100 Games

```
Reinforcement Learning Status:
        Completed 100 test episodes
        Average Rewards over testing: 1451.08
        Average Rewards for last 100 episodes: 1451.08
        Episode took 3354.56 seconds
Average Score: 1451.08
Scores:        1722, 1537, 1731, 1731, 1530, 1534, 1534, 1707, 101, 1530, 1531, 1329, 1326, 1731, 1530, 1729, 1737, 1525
, 1531, 1731, 1531, 1530, 1326, 1529, 1723, 1330, 1331, 1920, 1532, 1727, 1533, 1534, 1326, 1526, 1316, 1515, 1530, 1931
, 1530, 1519, 1729, 1334, 1323, 1712, 1537, 1944, 1730, 210, 1305, 1521, 1510, 1541, 1722, 708, 1522, 1528, 2131, 1518,
1931, 1318, 1942, 2115, 1719, 100, 1533, 1522, 443, 1323, 1531, 1531, 1329, 1531, 1731, 1931, 1331, 1326, 182, 1529, 133
1, 1331, 1532, 1531, 1339, 1331, 1529, 1941, 1518, 1530, 266, 1525, -43, 1716, 1529, 1535, 1343, 1537, 1731, 470, 1324,
1743
Win Rate:      91/100 (0.91)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
 Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
 Win, Win, Loss, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Loss, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Loss, Wi
n, Win, Win, Win, Win, Loss, Win, Win
PS G:\BINUS\Pacman> python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 10000 -n 10100 -l mediumClassi
c
```

6. 10000 Training – 1000 Games

```
09, 1535, 1525, 1726, 1931, 1536, 1525, 1336, 1735, 1537, 141, 1310, 1531, 1731, 1927, 1331, 1720, 1741, 1530, 470, 1520
, 1919, 1731, 482, 1728, 1936, 100, 1518, 1522, 1330, 1721, 1699, 1535, 1924, 1733, 162, 1518, 1337, 1531, 1731, 1334, 1
731, 1714, 1538, 1541, 1514, 1339, 1535, 1531, 1323, 1331, 1527, 1932, -51, 1522, -43, 1330, 1526, 1515, 1931, 1526, 152
7, 1531, 185
Win Rate:      887/1000 (0.89)
Record:        Win, Win, Loss, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Wi
n, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, W
in, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win,
 Win, Win, Win, Loss, Loss, Loss, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Loss, Loss, Loss, Win, Win, Win, Wi
n, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, W
in, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
 Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Win
```

V. Conclusion

In conclusion, training time, code optimization, and the size of the games played are several major factors in optimizing and maximizing the win rate. Even though the size of the learning is quite high, if the code is not optimized and the number of games played is short, the win rate of the game will be affected.

VI. Bibliography

Tutorial Website

http://ai.berkeley.edu/project_overview.html

Development Tools

https://www.python.org/download/releases/2.7/