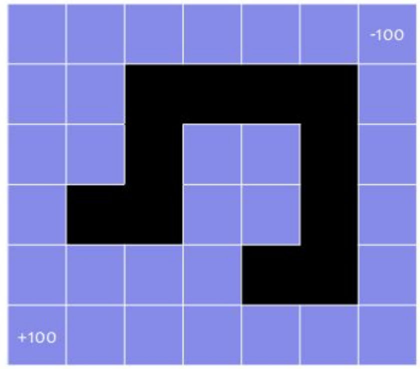
Reinforcement Learning

We extend the windy maze defined in P1 with probabilistic outcome after an action and two terminal states (left bottom and top right). It becomes a MDP problem. The maze map is shown in the following figure. However, we assume that the agent doesn't know either the reward function or the transition



model. The agent aims to run many trials in order to obtain Q-value for each (state, action) pair and the optimal action at each state.

Environment In your implementation, you need to simulate the windy maze environment: We assume that the wind comes from the south and the cost of one step for the agent is defined as follows: 1 for moving northward; 2 for moving westward or eastward; 3 for moving southward. The reward will be the negation of the reward. The agent can drift to the left or the right from the perspective of moving direction with probability 0.1. If the drifting direction is an obstacle, it will be bounced back to the original position. If the agent falls into any terminal state, it can't move out.

Reinforcement Learning In your implementation, you will generate many trials, each of which will result in a trajectory of (state, action, reward) tuple. The agent will use the ϵ-`Greedy` algorithm to choose an action at each state along each trajectory, where $\epsilon = 0.05$: the agent chooses a latest optimal action at each state with 95% and a random action with 5%. The initial state for each trial is chosen randomly and each trial will end at the goal state. Along each trajectory, the agent will use `Q-Learning` to update the Q-values. Since the reward function $R(s, a)$ here depends on both the state and the action taken at this state, the Q-value update equations should be revised accordingly.

$$N(s,a) \leftarrow N(s,a) + 1 \tag{1}$$

$$Q(s,a) \leftarrow Q(s,a) + \frac{1}{N_{s,a}}\Big(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big) \tag{2}$$

We choose $\gamma = 0.9$.

Testing and Outputs In your testing, generate 20,000 trials starting from a random open square. We initialize the Q-values for each trial at any state-action as 0 except for two terminal states with +100 and −100 respectively. After 20,000 trials, report the following three outcomes for each algorithm:

- the access frequency at each state-action $N_{s,a}$;
- the Q-value function at each state-action $Q(s,a)$;
- the optimal action at each state-action.

The expected outcome should look like as follows:

- **Table of $N(s, a)$:**

```
    1519              2062              3029              5021              8371              44
48        50      98        90      3974      94      2382      100     1389      129     2013      29
    1531              4474              70                101               148               29
    114               80                                                                                        55
85        99      5187      74          ####              ####              ####              ####          77        48
    6595              75                                                                                        4390
    115               112                                 5733              6770                                3405
129       123     2383      41          ####          112       133     1083      97          ####          697       1137
    8654              38                                  1853              121                                 2056
    131                                                   143               91                                  563
202       185         ####              ####          74        54      1385      35          ####          234       281
    9680                                                  4000              26                                  2612
    227               140               83                74                                                    185
221       221     9542      137     6461      81      4655      65          ####              ####          116       104
    17401             122               92                71                                                    4034
                      338               846               118               120               125               99
                  7462      105     6616      88      6712      75      6103      77      5195      76      4340      63
                      221               89                104               72                70                70
```

- **Table of $Q(s, a)$:**

```
    -5.2              -5.3              -5.7              -6.0              -6.5              -9.7
3.8       -1.6    6.9       0.8     8.1       -3.6    1.6       -6.5    -2.9      -8.0    -7.3      -77.7       -100
    24.7              17.0              -2.1              -6.0              -8.1              -11.7
    13.7              9.2                                                                                       -69.1
24.7      18.4    29.1      18.6        ####              ####              ####              ####          -14.5     -20.3
    40.3              28.4                                                                                      -12.2
    32.6              8.4                                 -6.0              -5.9                                -10.7
44.3      34.5    42.2      27.0        ####          -3.0      -4.8    8.3       -6.0        ####          -10.7     -10.7
    54.5              30.8                                17.4              -3.9                                -10.1
    48.5                                                  -0.3              -3.7                                -9.2
43.2      49.2        ####              ####          18.2      10.8    18.3      1.5         ####          -9.1      -9.0
    68.8                                                  31.4              5.4                                 -5.4
    61.8              56.3              46.8              21.3                                                  -7.0
72.0      64.3    71.1      53.0    59.3      39.3    46.0      32.5        ####              ####          -5.5      -5.4
    82.8              67.6              52.2              37.7                                                  1.2
                      60.2              45.5              33.3              20.7              7.0               -1.4
    +100          83.7      61.6    69.7      43.2    53.7      28.8    37.4      15.9    22.9      5.4     10.5      1.7
                      55.7              55.5              38.9              25.9              11.0              4.2
```

- **Table of the optimal action at each state:**

vvvv	vvvv	<<<<	<<<<	<<<<	<<<<	-100
vvvv	<<<<	####	####	####	####	vvvv
vvvv	<<<<	####	vvvv	<<<<	####	vvvv
vvvv	####	####	vvvv	<<<<	####	vvvv
vvvv	<<<<	<<<<	<<<<	####	####	vvvv
+100	<<<<	<<<<	<<<<	<<<<	<<<<	<<<<

where `<<<<`: moving westward; `^^^^`: moving northward; `>>>>`: moving eastward; `VVVV`: moving southward; $+100$ and -100: the terminal rewards.

E-Greedy

Exploitation vs. Exploration

- The algorithm was split into 2 main components
  - Exploitation (95%) vs. Exploration (5%) selection
  - Desired direction (80%) vs. Drift probability each side (10% per side)
- From the initial random choice, we select either exploitation or exploration:

Exploitation Path

```
int eGreedy(int x, int y, int epsilonPercent=5)
{
    //RNG check: 95% optimal action; 5% random action for epsilonPercent=5
    int choice = rng(100);
    int threshold = 100 - epsilonPercent;
    int action = 0;

    //Determine the action to be taken:
#pragma region Action determination
    //From 1-95: Optimal action
    if (choice <= threshold)
    {
        //Optimal action chosen
        //Check for single highest Q-Val and execute the action
        //If more than 1 highest Q-val, random between them
        float wQVal = 0.0;
        float nQVal = 0.0;
        float eQVal = 0.0;
        float sQVal = 0.0;
        //Check west q-val: Col-1
        wQVal = std::stof(qGrid[scaleGrid(x)][scaleGrid(y) - 1]);
        //Check north q-val: Row-1
        nQVal = std::stof(qGrid[scaleGrid(x) - 1][scaleGrid(y)]);
        //Check east q-val: Col+1
        eQVal = std::stof(qGrid[scaleGrid(x)][scaleGrid(y) + 1]);
        //Check south q-val: Row+1
        sQVal = std::stof(qGrid[scaleGrid(x) + 1][scaleGrid(y)]);

        std::cout << "e-Greedy: {Exploit}\n";
        //Action selected output: W/N/E/S
        action = maxQSelect(wQVal, nQVal, eQVal, sQVal);
    }
```

- In the event of selecting optimal routes, we select the action that has the highest Q-Value from the current grid position

- To select the proper action, and to account for potential equal Q-values that are also the highest, the maxQSelect function is used

```
int maxQSelect(float west, float north, float east, float south)
{
    int action = 0;
    bool wAction = false, nAction = false, eAction = false, sAction = false;
    float maxQVal = 0;
    int maxQCount = 0;  //If there are equal max Q, keep track of equally optimal choices
    maxQVal = std::max(west, std::max(north, std::max(east, south)));
    float qVal[4] = { west, north, east, south };

    for (int arrI = 0; arrI < 4; arrI++)
    {
        if (floatEquals(maxQVal, qVal[arrI]))
        {
            //Current direction is maxQVal
            maxQCount++;
            if (arrI == 0) wAction = true;
            else if (arrI == 1) nAction = true;
            else if (arrI == 2) eAction = true;
            else if (arrI == 3) sAction = true;
        }
    }

    //Action determination
    if (maxQCount == 1)
    {
        //Solo maxQVal action selected
        if (wAction == true) action = 1;
        else if (nAction == true) action = 2;
        else if (eAction == true) action = 3;
        else if (sAction == true) action = 4;
    }
    else if (maxQCount == 4)
    {
        action = rng(4);    //1 of 4 action selected
    }
```

- For cases with only 1 max Q-value, the choice is straightforward
- Similarly, for cases with all Q-value being equal, a random action is selected

```
else
{
    //from 2 or 3 specific actions to choose
    //Create rng buckets for each maxQVal action
    int bucket = rng(maxQCount * 100);
    //std::cout << "Bucket rng: " << bucket << " | ";

    //Each block of 100 numbers assigned to 1 action
    if (maxQCount == 2)
    {
        if (bucket <= 100)
        {
            if (wAction == true) action = 1;
            else if (wAction != true && nAction == true) action = 2;
            else if (wAction != true && nAction != true) action = 3;
        }
        else if (bucket > 100)
        {
            if (wAction == true)
            {
                if (nAction == true) action = 2;
                else if (eAction == true) action = 3;
                else if (sAction == true) action = 4;
            }
            else if (nAction == true)
            {
                if (eAction == true) action = 3;
                else if (sAction == true) action = 4;
            }
            else
            {
                action = 4;
            }
        }
    }
```

- For 2 max Q-Val selection, we select the respective actions
- The bucket will generate numbers from 1-200 in maxQCount == 2 case
    - 1-100 would select the lowest possible action (ie: action 1-West, if that was available; if action1-West was **not** the highest, it would check if action 2-North was available…)
    - 101-200 would select action 2 (similarly, the other possible action is the highest possible action available - action4-South if it was indeed the highest Q-Val, and checks lower actions if not)
- For reference:
    - Action 1: West
    - Action 2: North
    - Action 3: East
    - Action 4: South

```
        else if (maxQCount == 3)
        {
            if (bucket <= 100)
            {
                if (wAction == true) action = 1;
                else if (wAction != true && nAction == true) action = 2;
            }
            else if (bucket > 100 && bucket <= 200)
            {
                if (wAction == true)
                {
                    if (nAction == true) action = 2;
                    else if (nAction != true && eAction == true) action = 3;
                }
                else if (wAction != true) action = 3;
            }
            else
            {
                if (eAction == true) action = 3;
                else if (sAction == true) action = 4;
            }
        }
    }
    //std::cout << "Action: " << action << "\n";
    return action;
}
```

- Similarly, for 3 max Q-Val selection, the same process of selecting the proper respective actions as 2 max Q-Val case

Floating point precision

- Because floating points precision is an issue with "==" operator, a function with a cutoff resolution is used instead
- For practical purposes, differences smaller than the resolution ("epsilon") is considered equal

```
bool floatEquals(float a, float b)
{
    //Function to prevent floating point comparison using "==" to be inaccurate due to "fuzziness"
    float epsilon = 0.0000001;
    //For practical purposes, differences smaller than epsilon would be considered "equal"
    return std::abs(a - b) < epsilon;
}
```

Exploration Path

```
    //From 96-100: Random action
    else if (choice > threshold)
    {
        //Random action chosen (1 of 4 possible actions)
        int randChoice = rng(4);
        if (randChoice == 1) action = 1;        //West
        else if (randChoice == 2) action = 2;   //North
        else if (randChoice == 3) action = 3;   //East
        else if (randChoice == 4) action = 4;   //South
        std::cout << "e-Greedy: |Explore|\n";
    }
    std::cout << "Current position: " << x << "," << y <<"\n";
#pragma endregion Action determination
```

- A random action chosen with no regards to Q-Value

- From the action determination section, the “absolute” action is chosen that would be chosen in the event of 0 drift probability
- The next section will simply take the action into consideration and apply drift probability for the final “observed” action of the state-action pair

<u>Drift Probability</u>

```
#pragma region Drift
    //Once maxQSelect or rng selects the desired direction, account for drift:
    //80% to go towards desired action
    //10% to drift to each side

    int driftRng = rng(100);
    //If 1-80: Desired direction | 81-90: Drift to sideA | 91-100: Drift to sideB
    bool desiredDirection = false, driftA = false, driftB = false;
    if (driftRng <= 80) desiredDirection = true;
    else if (driftRng > 80 && driftRng <= 90) driftA = true;
    else if (driftRng > 90) driftB = true;

    std::cout << "Desired Action: " << action;
    switch (action)
    {
    case 1://Desired direction is West
        if (desiredDirection == true) action = 1;   //West
        else if (driftA == true) action = 2;        //North
        else if (driftB == true) action = 4;        //South
        break;
    case 2://Desired direction is North
        if (desiredDirection == true) action = 2;   //North
        else if (driftA == true) action = 1;        //West
        else if (driftB == true) action = 3;        //East
        break;
    case 3://Desired direction is East
        if (desiredDirection == true) action = 3;   //East
        else if (driftA == true) action = 2;        //North
        else if (driftB == true) action = 4;        //South
        break;
    case 4://Desired direction is South
        if (desiredDirection == true) action = 4;   //South
        else if (driftA == true) action = 1;        //West
        else if (driftB == true) action = 3;        //East
        break;
    }
#pragma endregion Drift
    if (desiredDirection != true) std::cout << " | Drift occured | ";
    std::cout << "| Actual Action: " << action << "\n";
    //Final resulting action (takes into account maxQVal, rng, and drift)
    //The true position that will end up happening
    return action;
}
```

- Action determination from above is the "absolute" direction we would taken if there was no drift possibility
- Desired direction (action) is taken at a 80% rate, 10% for each side
- The returned action from e-Greedy is the actual observed action of the state-action pair (s,a)

Q-Learning

```
std::pair<int,int> qLearning(int x, int y, int epsilonPercent=5, int actionOverride=0)
{
    int oldNVal = 0, newNVal = 0;
    float oldQVal = 0.0, newQVal = 0.0;
    float nextQVal = 0.0, deltaA = 0.0, deltaB = 0.0;
    float discount = 0.9, actionReward = 0.0;
    int action = 0;
    if (actionOverride != 0)
    {
        action = actionOverride;
    }
    else
    {
        action = eGreedy(x, y, epsilonPercent);
    }

    int nextRow = 0, nextCol = 0;
    float nextWQ = 0.0, nextNQ = 0.0, nextEQ = 0.0, nextSQ = 0.0;
    int i = 0, j = 0;
    //Input the state (position), eGreedy selects the action
    switch (action)
    {
    case 1: //West  (Col-1) | R(s,a) = -2
        i = scaleGrid(x), j = scaleGrid(y) - 1;
        actionReward = -2.0;
        if (grid[x][y - 1] == "####" || y - 1 < 0) { nextCol = y; }
        else { nextCol = y - 1; }
        nextRow = x;
        break;
    case 2: //North (Row-1) | R(s,a) = -1
        i = scaleGrid(x) - 1, j = scaleGrid(y);
        actionReward = -1.0;
        if (grid[x - 1][y] == "####" || x - 1 < 0) { nextRow = x; }
        else { nextRow = x - 1; }
        nextCol = y;
        break;
    case 3: //East  (Col+1) | R(s,a) = -2
        i = scaleGrid(x), j = scaleGrid(y) + 1;
        actionReward = -2.0;
        if (grid[x][y + 1] == "####" || y + 1 >= col) { nextCol = y; }
        else { nextCol = y + 1; }
        nextRow = x;
        break;
    case 4: //South (Row+1) | R(s,a) = -3
        i = scaleGrid(x) + 1, j = scaleGrid(y);
        actionReward = -3.0;
        if (grid[x + 1][y] == "####" || x + 1 >= row) { nextRow = x; }
        else { nextRow = x + 1; }
        nextCol = y;
        break;
    }
```

- To keep track of the data of the grid position's N(s,a) and Q(s,a) the grid is scaled up to a corresponding countGrid and qGrid and mapping the data to its respective positions
- Action rewards are assigned since e-Greedy provides the observed action which will be used for Q-Value update

Reference Grids:

	0	1	2	3	4	5	6
0							-100
1			##	##	##	##	
2			##			##	
3		##	##			##	
4					##	##	
5	100						

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0		X			X			X			X			X			X			X	
1	X	1,1	X	X	1,4	X	X	1,7	X	X	1,10	X	X	1,13	X	X	1,16	X	X	1,19	X
2		X			X			X			X			X			X			X	
3		X			X			X			X			X			X			X	
4	X	4,1	X	X	4,4	X	X	4,7	X	X	4,10	X	X	4,13	X	X	4,16	X	X	4,19	X
5		X			X			X			X			X			X			X	
6		X			X			X			X			X			X			X	
7	X	7,1	X	X	7,4	X	X	7,7	X	X	7,10	X	X	7,13	X	X	7,16	X	X	7,19	X
8		X			X			X			X			X			X			X	
9		X			X			X			X			X			X			X	
10	X	10,1	X	X	10,4	X	X	10,7	X	X	10,10	X	X	10,13	X	X	10,16	X	X	10,19	X
11		X			X			X			X			X			X			X	
12		X			X			X			X			X			X			X	
13	X	13,1	X	X	13,4	X	X	13,7	X	X	13,10	X	X	13,13	X	X	13,16	X	X	13,19	X
14		X			X			X			X			X			X			X	
15		X			X			X			X			X			X			X	
16	X	16,1	X	X	16,4	X	X	16,7	X	X	16,10	X	X	16,13	X	X	16,16	X	X	16,19	X
17		X			X			X			X			X			X			X	

```
    //Update the state-action
    std::pair<int, int> updatedPosition(nextRow, nextCol);

    oldNVal = std::stoi(countGrid[i][j]);
    newNVal = oldNVal + 1;

    oldQVal = std::stof(qGrid[i][j]);

    //Q-Learning update cases: Terminal or non-terminal states
    if (endState(updatedPosition))
    {
        //Goal
        if (grid[updatedPosition.first][updatedPosition.second] == "+100")
        {
            nextQVal = 100.0;
        }
        //Trap
        else if (grid[updatedPosition.first][updatedPosition.second] == "-100")
        {
            nextQVal = -100.0;
        }

    }
```

- Keep track of Q(s', a') in the event of terminal state
- This is more so for handling the way data is stored/retrieved from the larger scaled grid

```
    else //Not terminal state
    {
        nextWQ = std::stof(qGrid[scaleGrid(nextRow)][scaleGrid(nextCol) - 1]);
        nextNQ = std::stof(qGrid[scaleGrid(nextRow) - 1][scaleGrid(nextCol)]);
        nextEQ = std::stof(qGrid[scaleGrid(nextRow)][scaleGrid(nextCol) + 1]);
        nextSQ = std::stof(qGrid[scaleGrid(nextRow) + 1][scaleGrid(nextCol)]);

        nextQVal = std::max(nextWQ, std::max(nextNQ, std::max(nextEQ, nextSQ)));
    }
    deltaA = 1.0 / (float)newNVal; // 1/N(s,a)
    deltaB = actionReward + (discount * nextQVal) - oldQVal;    //R(s,a) + (gamma * maxNextQVal) - oldQVal
    newQVal = oldQVal + (deltaA*deltaB);

    //Store iteration calculations
    countGrid[i][j] = std::to_string(newNVal);
    qGrid[i][j] = std::to_string(newQVal);

    //std::cout << "Curent (x,y): " << x << "," << y << "\nAction: " << action << "\nNext (x,y): " << next
    return updatedPosition;
}
```

- N(s,a) has been stored using int, so (float) cast is necessary for proper delta calculations
- Once calculations are completed and the corresponding N(s,a) and Q(s,a) updated, return new position to begin the next action from the new state

```
e-Greedy: {Exploit}
Current position: 4,1
Desired Action: 1| Actual Action: 1
N(s,a):
          24                 182                 118                 77                  66                  29

    206    |    170   1124    |    15    840    |    14    632    |    11    430    |     6    207    |     4

        1192                 162                 110                 73                  53                  24

          27                  79                                                                                          3

    229    |    264    534    |     8            ####                ####                ####                ####       28    |    32

        1688                  76                                                                                        214

          35                  62                                     156                 291                                  8

    292    |    276    505    |     8            ####          79    |    93    319    |    44            ####       63    |    66

        2148                  62                                     442                  44                                416

          42                                                          22                  60                                 12

    378    |    283           ####               ####         113    |   104    297    |    11            ####      104    |    99

        2364                                                         851                  41                                618

          43                 208                  85                  19                                                     12

    501    |    499   1468    |    31    660    |    12    152    |   181            ####                ####      131    |   112

        3509                 199                  73                1211                                                    985

                             330                 341                 326                 211                 172                157

                  2535    |    31   2496    |    37   2580    |    37   1496    |    25   1270    |    18   1074    |    18

                             338                 378                 371                 222                 168                138
```

Zishen Teh

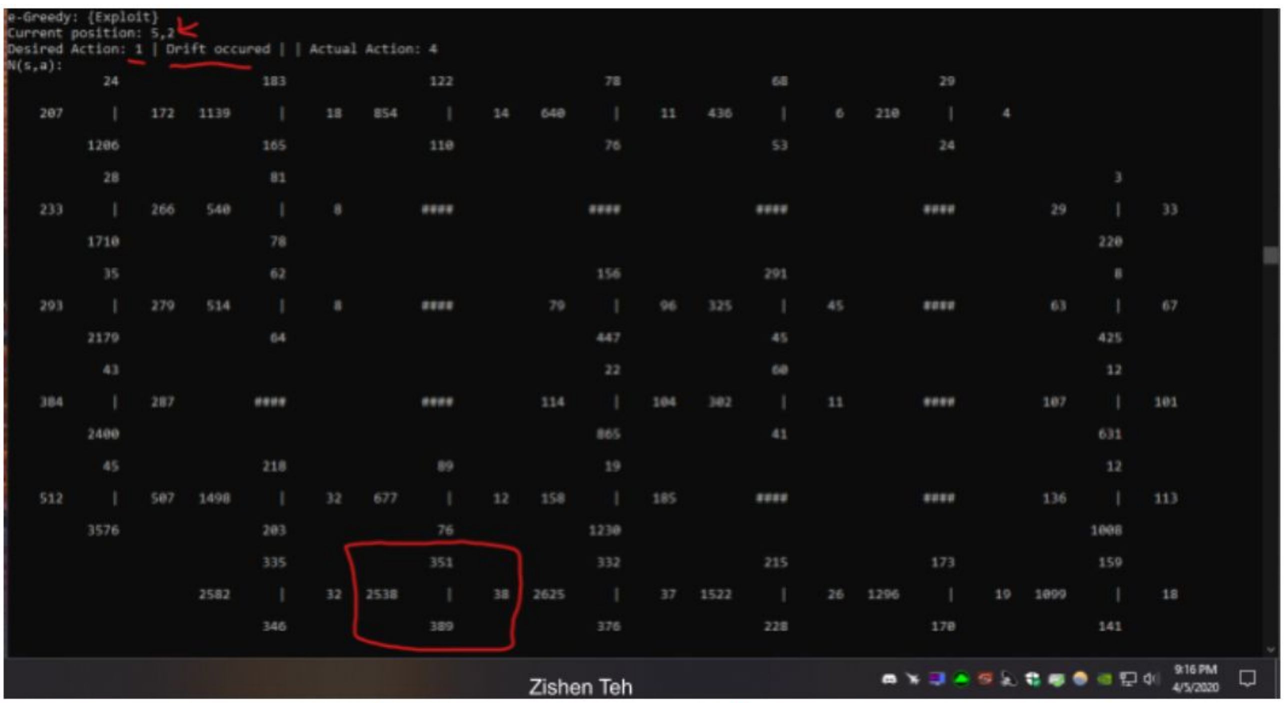
9:09 PM
4/5/2020

```
Q(s,a):
       39.33               35.08               29.67               24.8                20.42               16.48

  39.88      |  33.65  40.09      |  26.63  34.08      |  21.96  28.67      |  17.48   23.8      |  12.57  19.42      | -91.99              -100

       46.76               39.72               27.67               22.8                18.42               14.48

       39.53               35.08                                                                                           -90.99

  47.55      |  40.82  47.76      |  24.87             ####                ####                ####                ####      13.97      |  14.04

       55.29               47.66                                                                                               18.4

       47.34               41.99                                  -2.93               -0.91                                   13.49

  56.09      |  48.48  56.29      |  48.66             ####       23.63      |  11.13  28.39      |   -3.6             ####      19.06      |  19.08

       64.77               47.66                                  39.95               21.39                                   23.78

       55.91                                                      26.66               12.86                                   18.62

  65.59      |  65.53             ####                ####        40.1      |   31.8  41.37      |  17.21             ####      24.52      |  24.51

        75.3                                                       48.9               31.16                                   29.76

       65.19               67.67                  59              40.58                                                       23.55

  76.14      |  66.53   76.3      |     58  66.67      |  49.96   57.6      |  49.67             ####                ####      30.51      |  30.46

          87                76.2               66.48              57.73                                                        36.4

                           67.25               58.83              50.64               51.86               44.77               31.34

        +100                  88      |  65.24   77.2      |  57.09  67.48      |  49.43  58.73      |  41.94  50.86      |  35.21  43.77      |   37.4

                            76.2               66.48              57.73               49.86               42.77                36.4
```

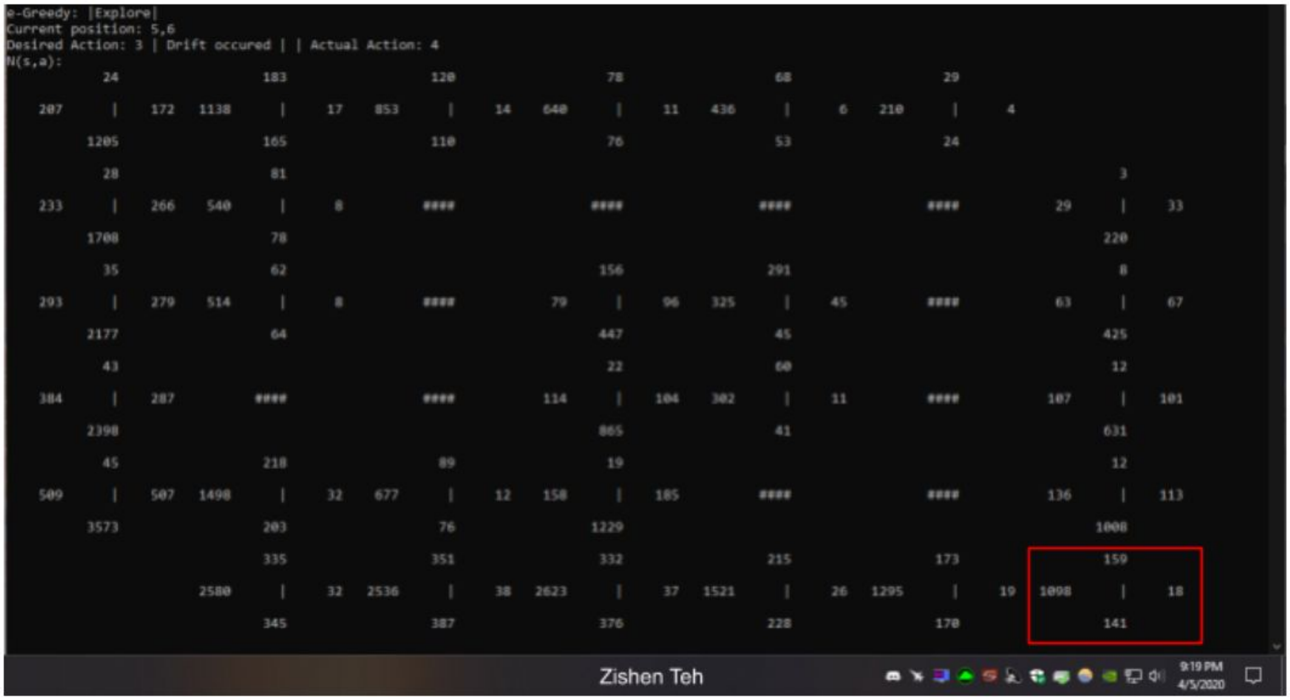
Zishen Teh

9:11 PM
4/5/2020

- Once the Q-Learning completes, the e-Greedy begins again from the new current position (4,0) - previously (4,1)
- E-greedy selects exploitation in this case, so the desired action is 4 (south)
- Because drift did not occur, the actual action taken is indeed south and reaches terminal state, which will end this specific trial run



- From position position (5,2), the desired action is 1 (West)
- However, drift occured and the action ended up drifting to the south instead



- In this case, the optimal action is action 1 (West), however, e-Greedy selects exploration and thus its desired action is action 3 (East)
- However, drift occured and ended up with action 4 (South) instead