Introduction To Reinforcement Learning
Project 2

Koray Emre Şenel - 150117037

Problem:

In this project, Q Learning algorithm is implemented.

Q-Learning:

For Q-learning this algorithm is used and shown which part of it, is done in the code with comments.

Algorithm for Q-learning

- *Initialize Q(s,a) arbitrarily*
- *Repeat for each episode*
  - *Initialize s;*
  - *Repeat for each step of episode*
    - *Choose a from s using policy derived from Q(e.g., ε-greedy)*
    - *Take action a, observe reward r, and next state s′*

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

    - $s \leftarrow s'$;
  - *Until s is terminal*
- *Until convergence occurs*

This is the main function. In this function initialization of start state and repeat for each episode part is done.

```java
public static void main(String[] args) {
    int j = 0;
    int counter ;
    int iteration = 100000000;
    int startState = 0;
    for (int i = 0; i < iteration; i++) {
        counter = episodes(startState, seeRoad: false);
```

In the episodes function, we have a loop with a flag condition where this flag is true until load, unload and returning to start state is done. The corresponding parts are commented in code.

```java
208    static int episodes( int startState, boolean seeRoad) {
209        int counter = 0;
210        int agentState = startState; //Initialize s
211        boolean flag = true;
212        isDone = false;
213        isLoaded = false;
214        boolean isLoadedSwap = true;
215        boolean isDoneSwap = true;
216        while (flag) { //Repeat for each state of episode
217            counter++;
218            int nextState;
219            int act ;
220            act = findAct(agentState); //Choose a from s using policy derived from epsilon greedy.
221            nextState = takeAction(act,agentState);//Take action and observer reward r and next state s'
222            if(nextState != agentState){
223                lastAct = act;
224                if(que.size() > 19){
225                    que.remove();
226                }
227                que.add(agentState);
228            }
```

The epsilon greedy method, a random number is generated between (0,1), if it is smaller than epsilon the random action is selected, else action with max Q value is selected.

```
static int findAct(int currentState){
    double rand = random.nextDouble();
    int act ;
    if (epsilon > rand) {
        if(currentState == loadState || currentState == unloadState){
            act = random.nextInt( bound: 5);
        }
        else act = random.nextInt( bound: 4);
    }
    else {
        act = actQ(currentState);
    }
    return act;
}
```

This part of code makes the Q table update for s,a.

```
double q = Q[agentState][act];//s states Q value
double max = maxQ(nextState);//Max actions Q value for s'
double value = q + alpha * (reward + gamma * max - q);//New value of Q(s,a)
setQ(value, agentState, act);//Update function
```

Next state set as agent state.

```
agentState = nextState;
```

Check for terminal state and reward.

```
if(isDone && nextState == startState){
    reward = 1000;
    flag = false;
}
```

In converge part, after 1000 episodes each 20 episode is taken and if variance of it is smaller than 0.1 execution terminated the result is printed.

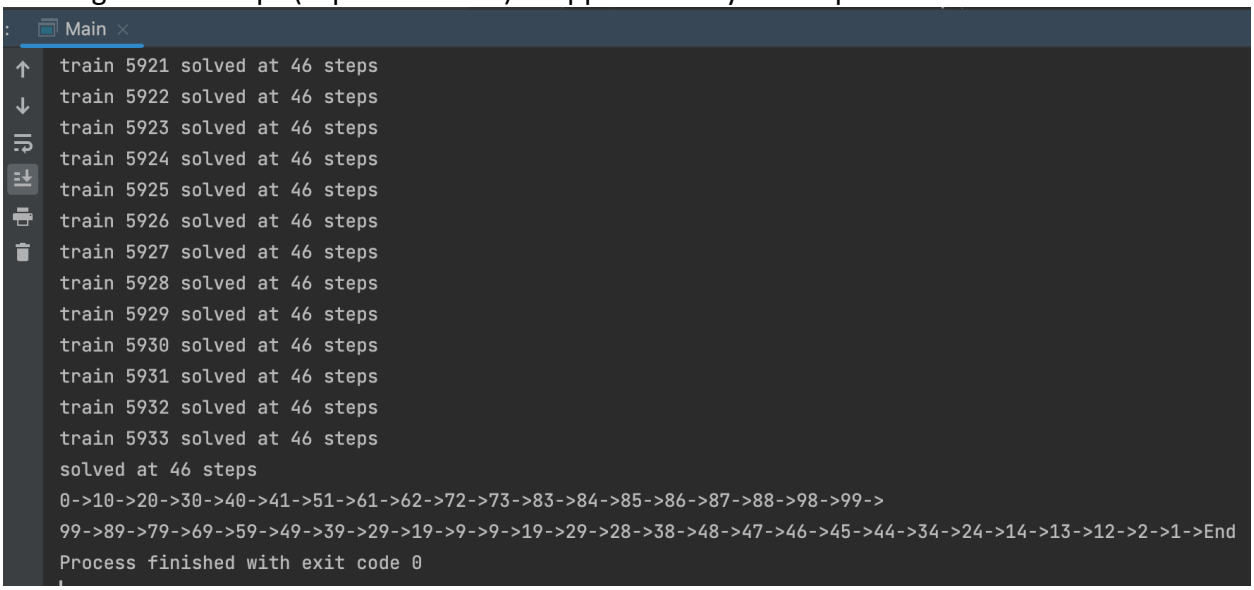
```
static boolean isLearned(int i, int counter) {
    int[] arr ;
    arr = stepQue.stream().mapToInt(Integer::intValue).toArray();
    double sumOfSample = 0;
    double variance;
    double sampleMean;
    for (int j = 0; j < i; j++)
        sumOfSample += arr[j];
    sampleMean = sumOfSample / i;
    double sumVari = 0;
    for (int j = 0; j < i; j++) {
        sumVari += Math.pow(((double) arr[j] - sampleMean), 2);
    }
    variance = sumVari / (double) (i - 1);
    if (variance < 0.1) {
        System.out.println("solved at " + counter + " steps");
        return true;
    } else return false;
}
```

What else used to solve this problem:

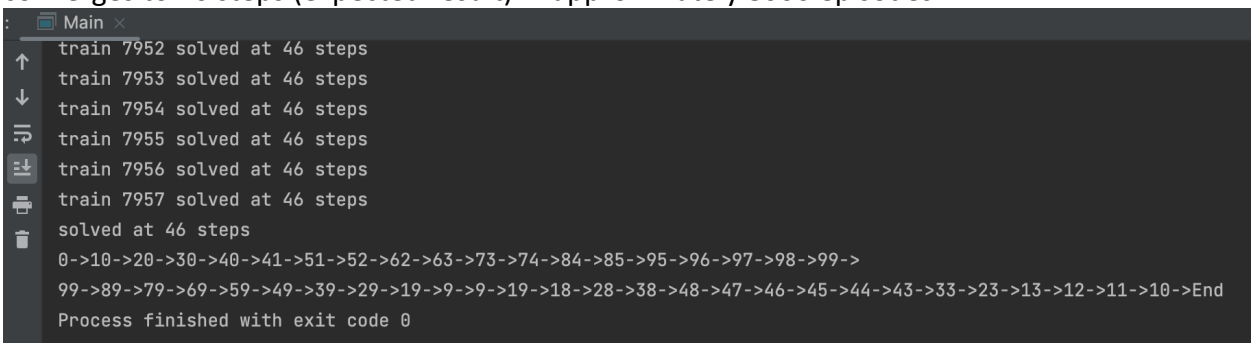
For encouraging agent to make load and unload operations, this operations gives 0 reward instead of -0.1. Also to check the correct road usage to load, looked the last 20 moves of the agent and check 3 of them have the correct road. If this road is not used, load is not working. For cross walls, the last successful action is stored, and used to determine which way it came to cross walls, action is done with that information.

Results of Q-learning:

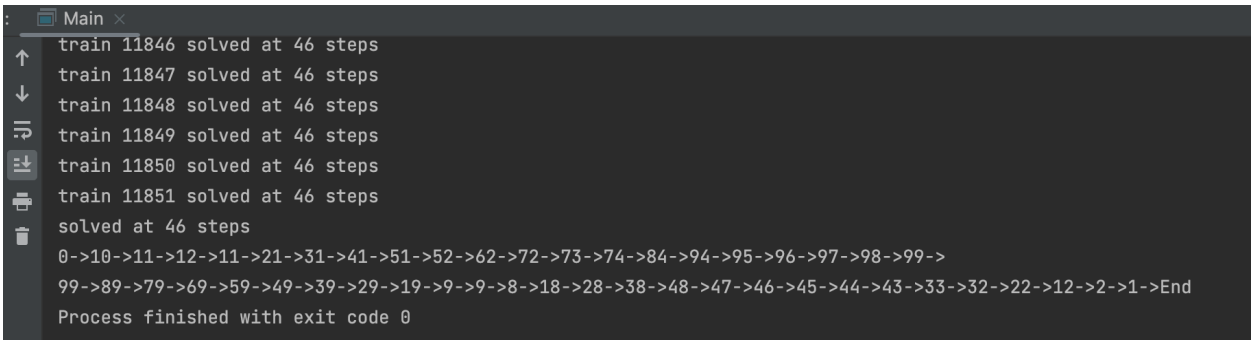
With the settings of alpha = 0.1, gamma = 0.9, epsilon = 0.3 (decreasing each episode) it converges to 46 steps (expected result) in approximately 6000 episodes.



With the settings of alpha = 0.1, gamma = 0.8, epsilon = 0.4 (decreasing each episode) it converges to 46 steps (expected result) in approximately 8000 episodes.



With the settings of alpha = 0.9, gamma = 0.3, epsilon = 0.6 (decreasing each episode) it converges to 46 steps (expected result) in approximately 12000 episodes.



With the settings of alpha = 0.1, gamma = 0.9, epsilon = 0.1 (decreasing each episode) it converges to 46 steps (expected result) in approximately 2000 episodes. This gives the best result in my experiments.