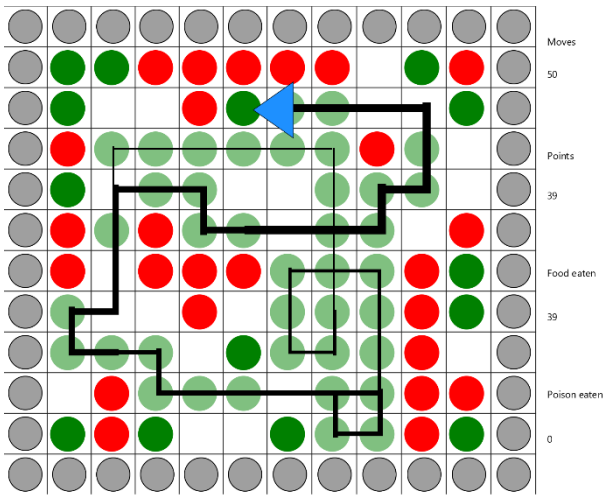
Report Bio-inspired AI project 1

Task 1:

I programmed this project using Java. Some important fields are agentOrientation {0,3}, agentCol, agentRow (to keep track of agent position and orientation), poisonTiles and foodTiles (both 2D lists with pointers to tiles with content). Most important methods are doNextMove, which does the next move based on orientation of the agent and picked move. PickNextMove decides whether to go left, forward, or right by a call to scoutMoves, which decides what move is the best by analyzing contents of possible moves. Also have a point counter which adds to total based on move and content of that next tile.



The agent picks moves as descried in the assignment document. Always avoid walls < do not eat poison unless the choice is to move into a wall < empty tiles < tiles with food. If there are several equal choices – go forward or pick randomly (if equals are left and right).

Avg score achieved over 1000 trials is 20 points.

Task 2:

So, the method train weights takes as input the active input neurons (moveWeights), the active output neuron (move) and the output value of the output neurons (outputWeights). All weights are stored in a 2D list (allWeights). The delta value is calculated by summing all the weights in the fasion described in the assignment, and checking what move is the correct according to the baseline agent (teacher move). If the move chosen by the supervised agent is the same one is added to the returned value. If not, we do not add 1 to the normalization.

```
private void trainWeights(ArrayList<Integer> moveWeights, int move, ArrayList<Double> outputWeights) {
    // TODO Auto-generated method stub

    int teacherMove = scoutMoves();
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            double newWeight = allWeights.get(i).get(moveWeights.get(j));
            double delta = getDelta(i+1,teacherMove, outputWeights);
            newWeight += LEARNING_RATE * delta;
            allWeights.get(i).set(moveWeights.get(j), newWeight);
        }
    }

}

private double getDelta(int move, int teacherMove, ArrayList<Double> outputWeights) {
    double sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += Math.exp(outputWeights.get(i));
    }
    if (teacherMove == move) {
        return -((double) Math.exp(outputWeights.get(move-1))/(double) sum) + (double) 1;
    } else {
        return -((double) Math.exp(outputWeights.get(move-1))/(double) sum);
    }
}
```

Figure 1: Calculation of delta for the supervised agent

Task 3:

The algorithm simulates a move and keeps track of the chosen output neuron value and the position of the agent in that state and the reward. Then it performs a new search for the best move in the next state, and

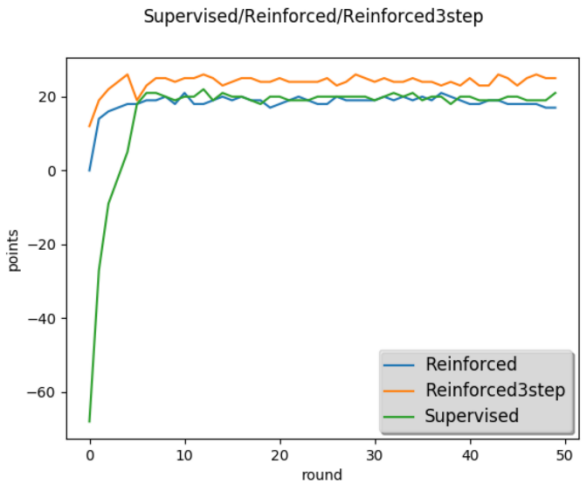
```
int move = findStrongestOutputNeuron(outputWeights.get(0),outputWeights.get(1),outputWeights.get(2));
double q = outputWeights.get(move-1);
int oldOrientation = getAgentOrientation();
int oldRow = getAgentRow();
int oldCol = getAgentCol();
int orientation = oldOrientation;
if (move == 1) {
    orientation = -1;
} else if (move == 2) {
    orientation = 0;
} else {
    orientation = 1;
}
orientation = setAndGetAgentOrientationTemp(orientation);
```

```
int reward = getPoints(agentRow, agentCol, true);
ArrayList<Integer> moves2 = identifyMoveContent();
ArrayList<Double> outputWeights2 = computeOutputWeights(moves2);


int move2 = findStrongestOutputNeuron(outputWeights2.get(0),outputWeights2.get(1),outputWeights2.get(2));
//lag en reversibel flytting av agenten
//hva skjer om den går inn i en vegg?
double delta = (double) reward + DISCOUNT_FACTOR*outputWeights2.get(move2-1) - q;
trainWeights(moves, move, outputWeights, delta);
agentCol = oldCol;
agentOrientation = oldOrientation;
agentRow = oldRow;
```

Figure 2 Calculation of delta for the reinforced agent

saves this value. Now it has all the components it needs to calculate the delta value. And the weights connected to the output neuron are trained. Before returning a move, the original state is reinstated.



Task 2,3,4 plot:

Each round consisting of 100 trials

Task 5 weights for reinforced NN:

```
Move left, x1:
[-2.9496437272058147, -1.6992523528772356, 2.2189176338262984, 3.665485180142302,
-1.3065617999685113, 0.9164836615957498, 0.8102418881622622, 0.8153429840960642,
0.24979123410946863, 0.30188165915058374, 0.31508975655424665, 0.368744084071335851
Move forward, x2:
[-0.19141920400464646, 0.329846343140764, 0.29836347587027723, -0.1936411537969776,
-4.900226794802421, -1.7301504144429802, 2.692950602060296, 4.180576068394395,
-0.47920921342256245, 0.13389495467863577, 0.24834542931522322, 0.3401182906381065]
Move right, x3:
[0.23120602120305794, 0.3215548786856845, 0.10989318487033502, 0.0424560250378687,
-0.8768449109738861, 1.1564052657519701, 1.0223813045231018, -0.596831549504233,
-2.960521126863548, -1.7255981769346236, 1.9349529884870722, 3.456276425107962]

Each weight col corresponds to input neuron [wall, poison, empty, food], whilst the
row corresponds to output neuron [left, forward, right].

Given input values on the form [leftContent, forwardContent, rightContent] with values
ranging from 0-3. Given the weights listed above me get the following output values
with inputs as follows:
```

Input, y	Moveleft= $\sum_{i=0}^{2} x1(y(i))+4i$	MoveForward= $\sum_{i=0}^{2} x2(y(i))+4i$	MoveRight= $\sum_{i=0}^{2} x3(y(i))+4i$	Selected a=max(L,F,R)
[0,0,2]	-3.9	-4.79	1.26	R
[1,3,2]	-0.57	4.74	1.63	F
[1,0,1]	-2.69	-4.45	-2.25	R
[3,3,3]	4.83	4.33	2.9	L
[2,3,0]	3.29	3.99	-3.39	F

```
By inspecting the move weights we can see that if the content of the cell
corresponding to the move is a wall, this weight is negative for all actions. We also
see that if the content is food, that weight is positive. This means that taking this
action in that state will be discouraged by the agent, which in terms gives us the
behavior we want (avoiding walls < eating poison < (go to empty spot) < eating food).
We see that the weights for moveForward have been trained more, this is because of the
case where several neurons are equally activated, in which case the moveForward action
wins.

The progression was steadily upwards. Both the supervised and reinforced agent did not
peak the baseline agent. Whilst the improved reinforced agent did perform better than
the others. This was as expected since the supervised agent cannot be better than its
teacher. The reinforced agent cannot be better than the baseline agent since the
baseline agent performs optimally based on its inputs, and the two agent have the same
inputs. The reinforced agent performs better because it has a bigger understanding of
its environment (due to better vision) than the other agents, and thus it outperforms
them all.
```