CPEN 502 Assignment-b: Reinforcement Learning (Look Up Table)

Ali Asgari Khoshouyeh (Student #24868739)

24. November 2021

1 Team Members

We are a team of three sharing the same code base.

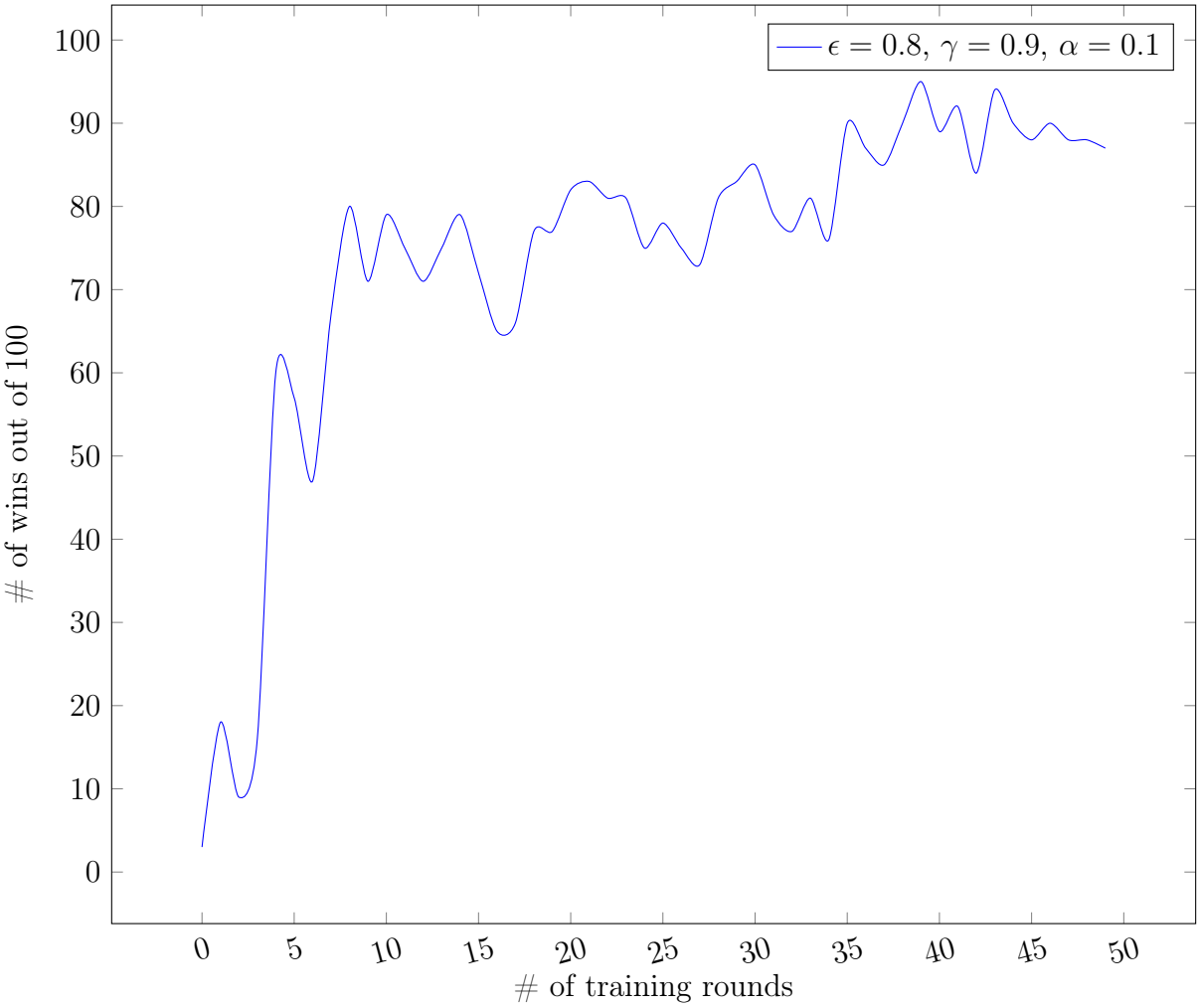
- Christina Sun
- Husna Kalim
- Ali Asgari Khoushouyeh

It is noteworthy to mention that close to the extended deadline we realized that our code is orders of magnitude slower on my teammates' machines. So we sharing the plot data too.

2 Q Learning Robot

(2) Once you have your robot working, measure its learning performance as follows:

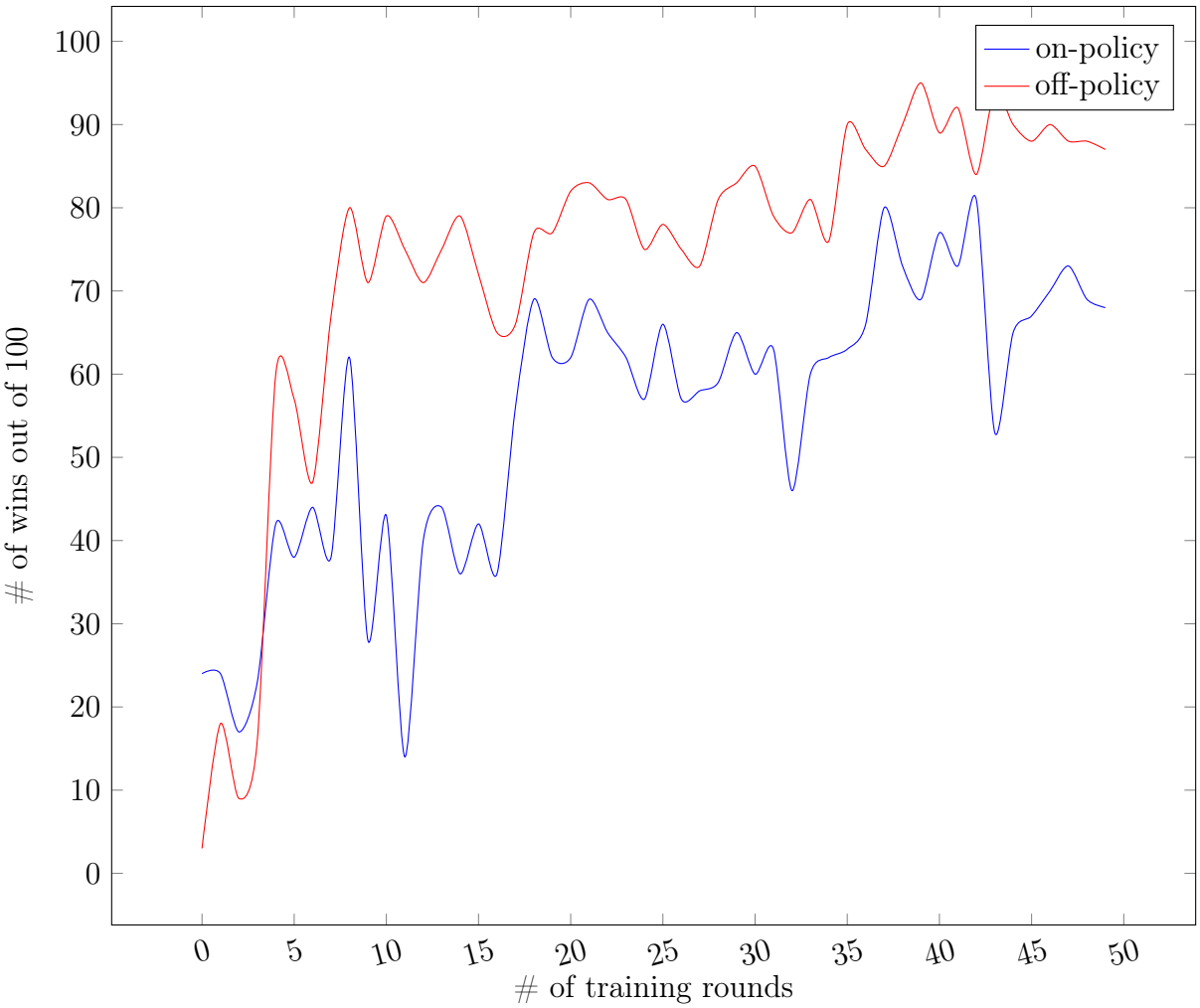
a) Draw a graph of a parameter that reflects a measure of progress of learning and comment on the convergence of learning of your robot.



2-a

We selected *Corners* robot as our opponent. We heuristically knew that it is a good strategy to always fire to defeat the enemy. As this is a simple thing to figure out, we observe that the robot converges pretty fast at a good win rate of 90/100. Please note that for obtaining the win ratio, through the report, we always use a robot that uses a trained LUT but works with $\epsilon = 0.05$ so that it actually exploits the trained LUT.

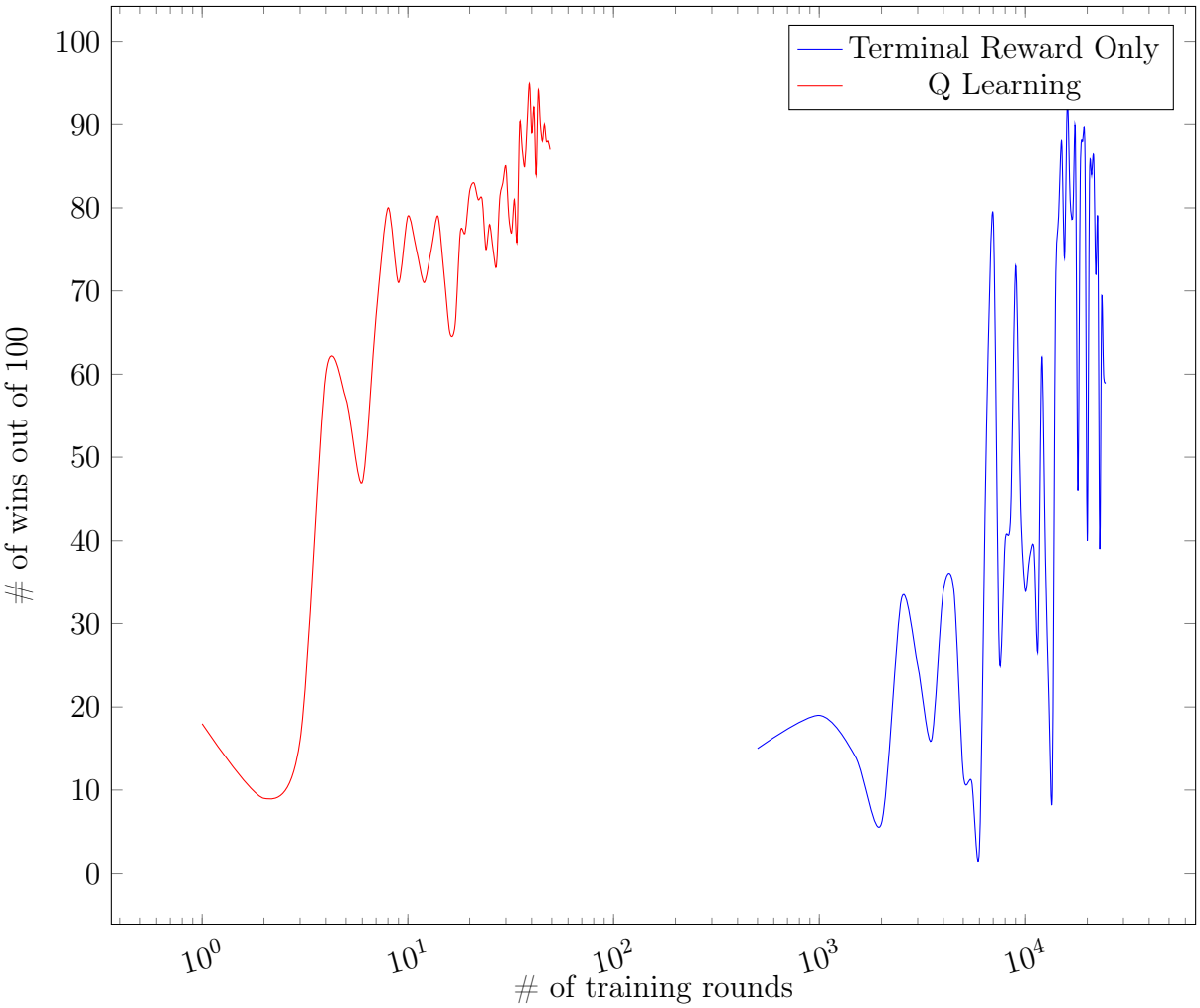
b) Using your robot, show a graph comparing the performance of your robot using on-policy learning vs off-policy learning.



2-b

As you can see the off-policy learning converges at a lower win rate when we train it with the same number of rounds. From the course content we know that the on-policy learning may result into a more conservative robot. Hereby the robot might not be as risk taking. We are coding the decrease in enemy's energy as the reward, so there might be cases where our robot prefers to run away not to get hit instead of firing and winning.

c) Implement a version of your robot that assumes only terminal rewards and show & compare its behaviour with one having intermediate rewards.



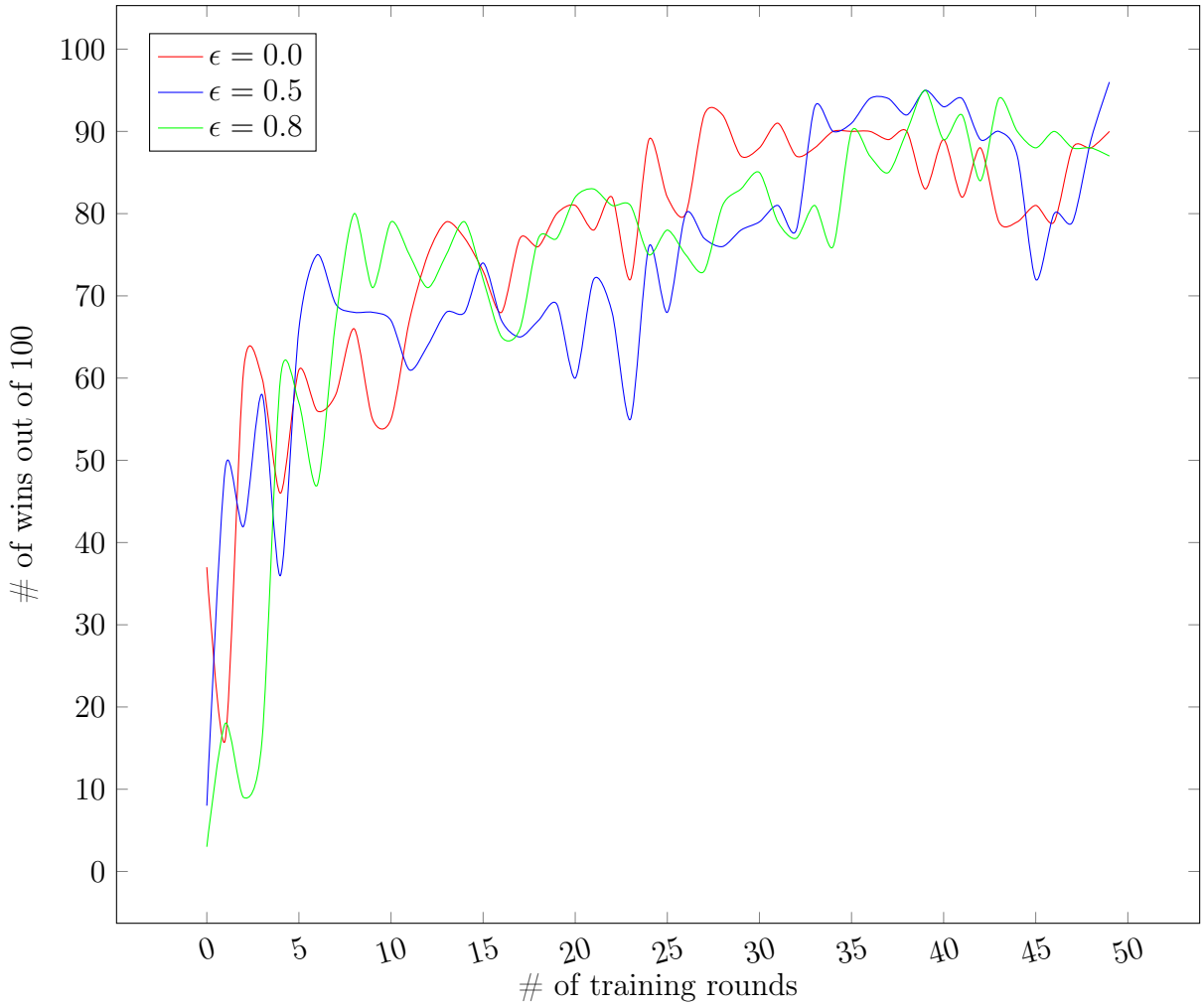
2-c

As you can see it takes 500x rounds to train the robot with only the terminal rewards. This is because the decrease in the enemy's energy is a really good approximate of actual winning and is early enough to provide our robot with adequate feedback. We coded +1 for winning and -1 reward for losing.

3 Role of ϵ

(3) This part is about exploration. While training via RL, the next move is selected randomly with probability ϵ and greedily with probability $1 - \epsilon$

a) Compare training performance using different values of e including no exploration at all. Provide graphs of the measured performance of your tank vs ϵ



3-a

As it can be seen from the graph, there is no significant difference between the performance or our robot with regard to different values for ϵ. We attribute this to the simplicity of the task of winning the *Corners* robot.

4 Lessons Learned

One of the challenges we faced during this assignment was that we wanted to code the reward of hitting bullets in a way that it is applied in the state where the bullet was fired. Because of the delay, it can take up to 3 turns for the bullet to hit the enemy and only then we can observe the reward.

In order to correspond the steps and the rewards more accurately first we slowed down taking actions. We took actions every three turn. It was good for training and actually solved our training issue and the robot learned the strategy of almost always firing but it was not good for the test time. It only took actions third of the time and it caused it to loose.

To solve that issue we added a time depth to our Q Learning algorithm. The only difference is that instead of using state-action pairs as the keys to the lookup table, we are using triplets of state-actions (including the two past state-actions). This actually makes the time visible to our robot and we observed that it can easily and quickly learn the correspondence of rewards and the actions even if the rewards are delayed for 3 steps.

Appendices

A Source Codes

```java
package autograd;

import jdk.jshell.spi.ExecutionControl;

public class Addition extends Operator {
    @Override
    public double evaluate(IVariable[] operands) {
        double result = 0.;
        for (IVariable operand :
                operands) {
            result += operand.evaluate();
        }
        return result;
    }

    @Override
    public void backwards(IVariable[] operands, IVariable[] sources, double
        gradient) throws ExecutionControl.NotImplementedException {
        for (IVariable o :
                operands) {
            o.backward(sources, gradient);
        }
    }
}
```

Listing 1: autograd/Addition.java

```java
package autograd;

import jdk.jshell.spi.ExecutionControl;

public class Exponentiation extends Operator {
    @Override
    public double evaluate(IVariable[] operands) {
        if (operands.length != 2) {
            throw new IllegalArgumentException("Exponentiation accepts 2
                arguments.");
        }
        return Math.pow(operands[0].evaluate(), operands[1].evaluate());
    }

    @Override
    public void backwards(IVariable[] operands, IVariable[] sources, double
        gradient) throws ExecutionControl.NotImplementedException {
        IVariable baseVariable = operands[0];
        var baseValue = baseVariable.evaluate();
        IVariable exponentVariable = operands[1];
        var exponentValue = exponentVariable.evaluate();
        if (exponentVariable.getParameters().length > 1) {
            throw new ExecutionControl.NotImplementedException("Back
                propagation to the exponent is not implemented.");
        }
```

```
23          var gradientToPropagate = Math.pow(gradient * baseValue *
                exponentValue, exponentValue - 1);
24          baseVariable.backward(sources, gradientToPropagate);
25      }
26  }
```

Listing 2: autograd/Exponentiation.java

```
1  package autograd;
2
3  public interface IInitializer {
4      double next();
5  }
```

Listing 3: autograd/IInitializer.java

```
1  package autograd;
2
3
4  import jdk.jshell.spi.ExecutionControl;
5
6  public interface IOperator {
7      IVariable apply(IVariable... operands);
8
9      double evaluate(IVariable[] operands);
10
11     void backwards(IVariable[] operands, IVariable[] sources, double
           gradient) throws ExecutionControl.NotImplementedException;
12 }
```

Listing 4: autograd/IOperator.java

```
1  package autograd;
2
3  import jdk.jshell.spi.ExecutionControl;
4
5  public interface IVariable {
6      double evaluate();
7
8      void backward(IVariable[] sources, double gradient) throws
           ExecutionControl.NotImplementedException;
9
10     Parameter[] getParameters();
11 }
```

Listing 5: autograd/IVariable.java

```
1  package autograd;
2
3  import jdk.jshell.spi.ExecutionControl;
4
5  public class Multiplication extends Operator {
6
7      @Override
8      public double evaluate(IVariable[] operands) {
9          double result = 1.;
10         for (IVariable operand :
11                 operands) {
```

```
            result *= operand.evaluate();
        }
        return result;
    }

    @Override
    public void backwards(IVariable[] operands, IVariable[] sources, double gradient) throws ExecutionControl.NotImplementedException {
        validateOperands(operands);
        var multiplier = operands[0];
        var multiplicand = operands[1];
        var multiplierValue = multiplier.evaluate();
        var multiplicandValue = multiplicand.evaluate();
        multiplier.backward(sources, gradient * multiplicandValue);
        multiplicand.backward(sources, gradient * multiplierValue);
    }
}
```

Listing 6: autograd/Multiplication.java

```
package autograd;

import jdk.jshell.spi.ExecutionControl;

public class Negation extends Operator {

    public Negation() {
        this.numberOfOperands = 1;
    }

    @Override
    public double evaluate(IVariable[] operands) {
        validateOperands(operands);
        return -operands[0].evaluate();
    }

    @Override
    public void backwards(IVariable[] operands, IVariable[] sources, double gradient) throws ExecutionControl.NotImplementedException {
        operands[0].backward(sources, -gradient);
    }
}
```

Listing 7: autograd/Negation.java

```
package autograd;

import jdk.jshell.spi.ExecutionControl;

import java.util.Arrays;
import java.util.HashSet;

public class Operation implements IVariable {
    private final IOperator operator;
    private final IVariable[] operands;

    public Operation(IOperator operator, IVariable... operands) {
        this.operator = operator;
```

```
14          this.operands = operands;
15      }
16
17      @Override
18      public double evaluate() {
19          return operator.evaluate(operands);
20      }
21
22      @Override
23      public void backward(IVariable[] sources, double gradient) throws
            ExecutionControl.NotImplementedException {
24          operator.backwards(operands, sources, gradient);
25      }
26
27
28      @Override
29      public Parameter[] getParameters() {
30          HashSet<Parameter> result = new HashSet<>();
31          for (IVariable o :
32                  this.operands) {
33              result.addAll(Arrays.asList(o.getParameters()));
34          }
35          return result.toArray(new Parameter[0]);
36      }
37
38      public IVariable[] getOperands() {
39          return operands;
40      }
41  }
```

Listing 8: autograd/Operation.java

```
1   package autograd;
2
3   public abstract class Operator implements IOperator {
4       protected Integer numberOfOperands;
5
6       public Operator() {
7           this.numberOfOperands = null;
8       }
9
10      @Override
11      public IVariable apply(IVariable... operands) {
12          return new Operation(this, operands);
13      }
14
15      protected void validateOperands(IVariable[] operands) {
16          if (this.numberOfOperands == null) {
17              return;
18          }
19          if (operands.length != this.numberOfOperands) {
20              throw new IllegalArgumentException(String.format("%s accepts
                    only one operand.", this.getClass().getName()));
21          }
22      }
23  }
```

Listing 9: autograd/Operator.java

```java
package autograd;

import java.util.Arrays;

public class Parameter implements IVariable {
    private double value;
    private double gradient;
    private boolean trainable;
    private int layer;

    public Parameter() {

    }

    public Parameter(double value) {
        this.value = value;
        trainable = true;
    }

    public Parameter(double value, boolean trainable) {
        this.value = value;
        this.trainable = trainable;
    }

    public static IVariable[] createTensor(double[] desired) {
        var result = new Parameter[desired.length];
        for (int i = 0; i < result.length; i++) {
            result[i] = new Parameter(desired[i]);
        }
        return result;
    }

    @Override
    public double evaluate() {
        return value;
    }

    @Override
    public void backward(IVariable[] sources, double gradient) {
        if (Arrays.stream(sources).anyMatch(x -> x == this)) {
            setGradient(gradient + getGradient());
        }
    }

    @Override
    public Parameter[] getParameters() {
        return new Parameter[]{this};
    }

    public double getValue() {
        return this.value;
    }

    public void setValue(double value) {
        this.value = value;
    }

```

```
58      public double getGradient() {
59          return gradient;
60      }
61
62      private void setGradient(double gradient) {
63          this.gradient = gradient;
64      }
65
66      public boolean isTrainable() {
67          return this.trainable;
68      }
69
70      public void zeroGradient() {
71          this.setGradient(0);
72      }
73
74      public int getLayer() {
75          return layer;
76      }
77
78      public void setLayer(int layer) {
79          this.layer = layer;
80      }
81  }
```

Listing 10: autograd/Parameter.java

```
1   package autograd;
2
3   import jdk.jshell.spi.ExecutionControl;
4
5   public class Sigmoid extends Operator {
6
7       public Sigmoid() {
8           this.numberOfOperands = 1;
9       }
10
11      @Override
12      public double evaluate(IVariable[] operands) {
13          if (operands.length != 1) {
14              throw new IllegalArgumentException("Sigmoid operator only
                    accepts one operand");
15          }
16          return 1. / (1 + Math.exp(-operands[0].evaluate()));
17      }
18
19      @Override
20      public void backwards(IVariable[] operands, IVariable[] sources, double
            gradient) throws ExecutionControl.NotImplementedException {
21          validateOperands(operands);
22          var x = operands[0];
23          var y = evaluate(operands);
24          x.backward(sources, gradient * y * (1 - y));
25      }
26  }
```

Listing 11: autograd/Sigmoid.java

```java
package autograd;

import java.util.Random;

public class UniformInitializer implements IInitializer {

    double a;
    double b;
    Random random;

    public UniformInitializer(double a, double b) {
        this.a = a;
        this.b = b;
        this.random = new Random();
    }

    @Override
    public double next() {
        return random.nextDouble() * (b - a) + a;
    }
}
```

Listing 12: autograd/UniformInitializer.java

```java
package dataset;

public class BinaryToBipolarWrapper implements IDataSet {

    IDataSet binaryDataSet;

    public BinaryToBipolarWrapper(IDataSet binaryDataSet) {
        this.binaryDataSet = binaryDataSet;
    }

    @Override
    public DataPoint next() {
        DataPoint result = binaryDataSet.next();
        if (result == null) return null;
        double[] x = result.getX().clone();
        double[] y = result.getY().clone();
        for (int i = 0; i < x.length; i++) {
            x[i] = 2 * x[i] - 1;
        }
        for (int i = 0; i < y.length; i++) {
            y[i] = 2 * y[i] - 1;
        }
        return new DataPoint(x, y);
    }

    @Override
    public void reset() {
        binaryDataSet.reset();
    }
}
```

Listing 13: dataset/BinaryToBipolarWrapper.java

```java
package dataset;
```

```
2
3  public class DataPoint {
4      private final double[] x;
5      private final double[] y;
6
7      public DataPoint(double[] x, double[] y) {
8          this.x = x;
9          this.y = y;
10     }
11
12     public double[] getY() {
13         return y;
14     }
15
16     public double[] getX() {
17         return x;
18     }
19 }
```

Listing 14: dataset/DataPoint.java

```
1  package dataset;
2
3  public interface IDataSet {
4      DataPoint next();
5
6      void reset();
7  }
```

Listing 15: dataset/IDataSet.java

```
1  package dataset;
2
3  public class XORBinaryDataSet implements IDataSet {
4
5      protected double[][] x;
6      protected double[] y;
7      private int index;
8
9      public XORBinaryDataSet() {
10         index = 0;
11         x = new double[][]{
12                 {0., 0.},
13                 {0., 1.},
14                 {1., 0.},
15                 {1., 1.},
16         };
17         y = new double[]{
18                 0.,
19                 1.,
20                 1.,
21                 0.,
22         };
23     }
24
25     @Override
26     public DataPoint next() {
27         if (index < x.length) {
```

```
            var result = new DataPoint(x[index], new double[]{y[index]});
            index++;
            return result;
        }
        return null;
    }

    @Override
    public void reset() {
        index = 0;
    }
}
```

Listing 16: dataset/XORBinaryDataSet.java

```
package fa;

import representation.IRepresentable;

import java.io.FileNotFoundException;
import java.io.IOException;

public interface IFunctionApproximation {
    void train(IRepresentable input, double[] output);
    double[] eval(IRepresentable input);

    void save() throws IOException;

    void load() throws IOException, ClassNotFoundException;
}
```

Listing 17: fa/IFunctionApproximation.java

```
package fa;

import representation.IRepresentable;

import java.io.*;
import java.util.HashMap;

public class LUT implements IFunctionApproximation {

    private final String filePath;
    int distance_level = 3;
    int robot_energy_level = 3;
    int enemy_energy_level = 3;
    int position_level = 48;
    HashMap StateMap = new HashMap(distance_level * robot_energy_level *
            enemy_energy_level * position_level);
    boolean readOnly;


    public LUT(String filePath, boolean readOnly) {
        this.filePath = filePath;
        this.readOnly = readOnly;
    }

    public void save(File argFile) {
```

```
26
27      }
28
29
30      public void load(String argFileName) throws IOException {
31
32      }
33
34      //LEFT HERE: finish the two methods below for implementation tomorrow
            morning.
35      // for each unique state vector, generate its key and match it with a
            state object that contains 5 actions
36
37      public void initialiseLUT() {
38          for (int distance = 0; distance < distance_level; distance++) {
39              for (int robot_energy = 0; robot_energy < robot_energy_level;
                    robot_energy++) {
40                  for (int enemy_energy = 0; enemy_energy <
                        enemy_energy_level; enemy_energy++) {
41                      for (int position = 0; position < position_level;
                            position++) {
42                          //double[] state_vector = {distance, robot_energy,
                                enemy_energy, position};
43                          //double key = indexFor(state_vector);
44  //                          State newState = new State(distance, robot_energy
    , enemy_energy, position);
45                          // Q values are automatically set to 0 by default
46  //                          newState.addAll(); // add all actions for each
    state?
47
48  //                          StateMap.put(newState, newState);
49
50                      }
51                  }
52              }
53          }
54      }
55
56
57
58      public double train(double[] X, double argValue) {
59
60
61
62
63          return 0;
64      }
65
66      @Override
67      public void train(IRepresentable input, double[] output) {
68          if (readOnly) {return;}
69          double[] repr = input.toVector();
70          System.out.print("train " + input + " to " + output[0]);
71          System.out.println();
72          this.StateMap.put(input, output);
73      }
74
75      @Override
```

```
76      public double[] eval(IRepresentable input) {
77          return (double[]) StateMap.getOrDefault(input, new double[]{ 0 });
78      }
79
80      @Override
81      public void save() throws IOException {
82          if (readOnly) return;
83          new ObjectOutputStream(new FileOutputStream(this.filePath)).
                writeObject(StateMap);
84      }
85
86      @Override
87      public void load() throws IOException, ClassNotFoundException {
88          this.StateMap = (HashMap) new ObjectInputStream(new FileInputStream
                (this.filePath)).readObject();
89      }
90
91      public int getSize() {
92          return this.StateMap.size();
93      }
94  }
```

Listing 18: fa/LUT.java

```
1   package nn;
2
3   import autograd.IVariable;
4   import autograd.Parameter;
5
6   public class BipolarSigmoid implements ILayer {
7
8       @Override
9       public IVariable[] apply(IVariable[] input) {
10          var sigmoid = new autograd.Sigmoid();
11          var scalar = new Parameter(2, false);
12          var constant = new Parameter(-1, false);
13          var addition = new autograd.Addition();
14          var multiplication = new autograd.Multiplication();
15          var result = new IVariable[input.length];
16          for (int i = 0; i < input.length; i++) {
17              result[i] = addition.apply(
18                      multiplication.apply(
19                              scalar,
20                              sigmoid.apply(input[i])),
21                      constant
22              );
23          }
24          return result;
25      }
26  }
```

Listing 19: nn/BipolarSigmoid.java

```
1   package nn;
2
3   import java.util.ArrayList;
4
5   public class ConvergenceCollector implements IFitCallback {
```

```
    ArrayList<Double> loss;

    public ConvergenceCollector() {
        this.loss = new ArrayList<>();
    }

    @Override
    public void collect(int epoch, double loss) {
        this.loss.add(loss);
    }

    public int getEpochs() {
        return loss.size();
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < loss.size(); i++) {
            sb.append(+i + " " + loss.get(i) + "\n");
        }
        return sb.toString();
    }
}
```

Listing 20: nn/ConvergenceCollector.java

```
package nn;

import autograd.IInitializer;
import autograd.IVariable;
import autograd.Parameter;
import autograd.UniformInitializer;

public class Factory {
    public static Model createNeuralNetwork(int[] sizes, ILayer activation,
        IInitializer initializer) {
        if (sizes.length < 2) {
            throw new IllegalArgumentException("Sizes must at least contain
                2 integers for the first and the second layer.");
        }
        var inputs = new Parameter[sizes[0]];
        for (int i = 0; i < inputs.length; i++) {
            inputs[i] = new Parameter(initializer.next());
        }
        IVariable[] lastLayerOutput = inputs;
        for (int i = 1; i < sizes.length; i++) {
            lastLayerOutput = new Linear(sizes[i - 1], sizes[i],
                initializer).apply(lastLayerOutput);
            lastLayerOutput = activation.apply(lastLayerOutput);
        }
        return new Model(inputs, lastLayerOutput);
    }

    public static Model createNeuralNetwork(int[] sizes, ILayer activation)
        {
        return createNeuralNetwork(sizes, activation, new
            UniformInitializer(-0.5, 0.5));
```

```
    }
}
```

Listing 21: nn/Factory.java

```
package nn;

public interface IFitCallback {
    void collect(int epoch, double loss);
}
```

Listing 22: nn/IFitCallback.java

```
package nn;

import autograd.IVariable;

public interface ILayer {
    IVariable[] apply(IVariable[] input);
}
```

Listing 23: nn/ILayer.java

```
package nn;

import autograd.*;

public class Linear implements ILayer {
    private final IVariable[][] weight;
    private final IVariable[] bias;

    public Linear(int inFeatures, int outFeatures, IInitializer initializer
        ) {
        this.weight = new Parameter[outFeatures][inFeatures];
        this.bias = new Parameter[outFeatures];
        for (int i = 0; i < outFeatures; i++) {
            for (int j = 0; j < inFeatures; j++) {
                this.weight[i][j] = new Parameter(initializer.next());
            }
            this.bias[i] = new Parameter(initializer.next());
        }
    }

    @Override
    public IVariable[] apply(IVariable[] input) {
        var result = new IVariable[this.weight.length];
        for (int i = 0; i < this.weight.length; i++) {
            int inputSize = this.weight[i].length;
            IVariable[] muls = new IVariable[inputSize + 1];
            for (int j = 0; j < inputSize; j++) {
                muls[j] = new Multiplication().apply(this.weight[i][j],
                    input[j]);
            }
            muls[inputSize] = this.bias[i];
            result[i] = new Addition().apply(muls);
        }
        return result;
    }
```

```
34
35
36      private int getWidth() {
37          return this.weight.length;
38      }
39  }
```

Listing 24: nn/Linear.java

```
1   package nn;
2
3   import autograd.*;
4   import jdk.jshell.spi.ExecutionControl;
5   import optimization.ILoss;
6
7   public class MinimumSquaredError implements IVariable, ILoss {
8
9       private final IVariable operation;
10      private final Parameter[] desired;
11
12      public MinimumSquaredError(IVariable[] output) {
13          var negation = new Negation();
14          var addition = new Addition();
15          var multiplication = new Multiplication();
16          var exponentiation = new Exponentiation();
17          Parameter two = new Parameter(2, false);
18          Parameter half = new Parameter(0.5, false);
19          int length = output.length;
20          desired = new Parameter[output.length];
21          var summationTerms = new IVariable[length];
22          for (int i = 0; i < length; i++) {
23              desired[i] = new Parameter();
24              summationTerms[i] = exponentiation.apply(
25                      addition.apply(output[i], negation.apply(desired[i])),
26                      two
27              );
28          }
29          this.operation = multiplication.apply(addition.apply(summationTerms
                ), half);
30      }
31
32      @Override
33      public double evaluate() {
34          return operation.evaluate();
35      }
36
37      @Override
38      public void backward(IVariable[] sources, double gradient) throws
            ExecutionControl.NotImplementedException {
39          operation.backward(sources, gradient);
40      }
41
42      @Override
43      public Parameter[] getParameters() {
44          return this.operation.getParameters();
45      }
46
47      @Override
```

```
48      public void setDesired(double[] desired) {
49          for (int i = 0; i < this.desired.length; i++) {
50              this.desired[i].setValue(desired[i]);
51          }
52      }
53  }
```

Listing 25: nn/MinimumSquaredError.java

```
1   package nn;
2
3   import autograd.IVariable;
4   import autograd.Operation;
5   import autograd.Parameter;
6   import dataset.DataPoint;
7   import dataset.IDataSet;
8   import jdk.jshell.spi.ExecutionControl;
9   import optimization.ILoss;
10  import optimization.IOptimizer;
11
12  import java.util.Arrays;
13  import java.util.HashSet;
14  import java.util.List;
15  import java.util.Map;
16  import java.util.stream.Collectors;
17
18  public class Model {
19      private final Parameter[] input;
20      private final IVariable[] output;
21
22      public Model(Parameter[] input, IVariable[] output) {
23          this.input = input;
24          this.output = output;
25      }
26
27
28      public double[] evaluate(double[] input) {
29          var result = new double[output.length];
30          for (int i = 0; i < input.length; i++) {
31              this.input[i].setValue(input[i]);
32          }
33          for (int i = 0; i < output.length; i++) {
34              result[i] = output[i].evaluate();
35          }
36          return result;
37      }
38
39      public Parameter[] getParameters() {
40          HashSet<Parameter> result = new HashSet<>();
41          for (IVariable o :
42                  this.output) {
43              result.addAll(Arrays.asList(o.getParameters()));
44          }
45          return result.toArray(new Parameter[0]);
46      }
47
48      public Parameter[] getTrainableParameters() {
49          var results = new HashSet<Parameter>();
```

```
50          for (Parameter p :
51                  getParameters()) {
52              if (p.isTrainable()) {
53                  results.add(p);
54              }
55          }
56          for (Parameter p : input) {
57              results.remove(p);
58          }
59
60          return results.toArray(new Parameter[0]);
61      }
62
63      public IVariable[] getOutput() {
64          return output;
65      }
66
67      public double fit(IDataSet dataSet, IOptimizer optimizer, ILoss loss,
            int epochs, double lossLimit) throws ExecutionControl.
            NotImplementedException {
68          return fit(dataSet, optimizer, loss, epochs, lossLimit, (epoch, l)
                -> {
69          });
70      }
71
72      public double fit(IDataSet dataSet, IOptimizer optimizer, ILoss loss,
            int epochs, double lossLimit, IFitCallback callback) throws
            ExecutionControl.NotImplementedException {
73          var parameters = getTrainableParameters();
74          Map<Integer, List<Parameter>> layeredParameters = layerParameters(
                parameters);
75          if (epochs < 1) {
76              throw new IllegalArgumentException("At least one epochs
                    required.");
77          }
78          double totalLoss = 0;
79          for (int i = 0; i < epochs; i++) {
80              totalLoss = 0;
81              dataSet.reset();
82              DataPoint dataPoint;
83              while ((dataPoint = dataSet.next()) != null) {
84                  setInput(dataPoint.getX());
85                  loss.setDesired(dataPoint.getY());
86                  totalLoss += loss.evaluate();
87                  for (Integer j : layeredParameters.keySet().stream().sorted
                        ().collect(Collectors.toList())) {
88                      Parameter[] layerParameters = layeredParameters.get(j).
                            toArray(new Parameter[0]);
89                      loss.backward(layerParameters, 1.);
90                      optimizer.update(layerParameters);
91                  }
92              }
93              callback.collect(i, totalLoss);
94              if (totalLoss < lossLimit) {
95                  break;
96              }
97          }
98          return totalLoss;
```

```
    }

    private Map<Integer, List<Parameter>> layerParameters(Parameter[]
        parameters) {
        setLayers(getOutput(), 0);
        return Arrays.stream(parameters).collect(Collectors.groupingBy(
            Parameter::getLayer));

    }

    private void setLayers(IVariable[] outputs, int layer) {
        if (outputs.length == 0) return;
        HashSet<IVariable> nextOutput = new HashSet<>();
        for (IVariable i : outputs) {
            if (i instanceof Parameter) {
                ((Parameter) i).setLayer(layer);
            }
            if (i instanceof Operation) {
                nextOutput.addAll(Arrays.asList(((Operation) i).getOperands
                    ()));
            }
        }
        setLayers(nextOutput.toArray(new IVariable[0]), layer + 1);
    }

    private void setInput(double[] x) {
        for (int i = 0; i < input.length; i++) {
            input[i].setValue(x[i]);
        }
    }
}
```

Listing 26: nn/Model.java

```
package nn;

import autograd.IVariable;

public class Sigmoid implements ILayer {

    @Override
    public IVariable[] apply(IVariable[] input) {
        var operator = new autograd.Sigmoid();
        var result = new IVariable[input.length];
        for (int i = 0; i < input.length; i++) {
            result[i] = operator.apply(input[i]);
        }
        return result;
    }
}
```

Listing 27: nn/Sigmoid.java

```
package optimization;

import autograd.Parameter;

import java.util.HashMap;
```

```
6
7  public class GradientDescent implements IOptimizer {
8
9      private final HashMap<Parameter, Double> lastDelta;
10     private final double learningRate;
11     private final double momentum;
12
13     public GradientDescent(double learningRate, double momentum) {
14         this.lastDelta = new HashMap<>();
15         this.learningRate = learningRate;
16         this.momentum = momentum;
17     }
18
19     @Override
20     public void update(Parameter[] parameters) {
21         for (Parameter p :
22                 parameters) {
23             double delta = -p.getGradient() * learningRate + momentum *
                   lastDelta.getOrDefault(p, 0.);
24             p.setValue(p.getValue() + delta);
25             p.zeroGradient();
26             lastDelta.put(p, delta);
27         }
28     }
29 }
```

Listing 28: optimization/GradientDescent.java

```
1  package optimization;
2
3  import autograd.IVariable;
4
5  public interface ILoss extends IVariable {
6      void setDesired(double[] desired);
7  }
```

Listing 29: optimization/ILoss.java

```
1  package optimization;
2
3  import autograd.Parameter;
4
5  public interface IOptimizer {
6      void update(Parameter[] parameters);
7  }
```

Listing 30: optimization/IOptimizer.java

```
1  package policy;
2
3  import representation.IState;
4  import representation.States;
5
6  public class EnergyReward implements IPolicy {
7      @Override
8      public double getReward(IState run, IState last) {
9          // should we give rewards on one state or on the change of last two
               states?
```

```
        States states = (States) run;
        States lastStates = (States) last;
//        return states.getMyEnergy() - states.getEnemyEnergy();

        return lastStates.getEnemyEnergy() - states.getEnemyEnergy();
    }
}
```

Listing 31: policy/EnergyReward.java

```
package policy;

import representation.IState;
import representation.States;

public class EnergyRewardTerminal implements IPolicy {
    @Override
    public double getReward(IState run, IState last) {
        // should we give rewards on one state or on the change of last two
            states?
        States states = (States) run;
        States lastStates = (States) last;
//        return states.getMyEnergy() - states.getEnemyEnergy();

        if (states.getEnemyEnergy() == 0)
            return 1;
        if (states.getMyEnergy() == 0)
            return -1;
        return 0;
    }
}
```

Listing 32: policy/EnergyRewardTerminal.java

```
package policy;

import representation.Coordinates;
import representation.IState;

public class GoTopRight implements IPolicy {
    @Override
    public double getReward(IState run, IState dummy) {
        Coordinates coordinates = (Coordinates) run;
        var x = coordinates.getX();
        var y = coordinates.getY();
        if (x == 7 && y == 5) {
            return 1;
        }
        return 0;
    }
}
```

Listing 33: policy/GoTopRight.java

```
package policy;

import representation.IState;

```

```java
public interface IPolicy {
    double getReward(IState currentState, IState lastState);
}
```

Listing 34: policy/IPolicy.java

```java
package representation;

public class Action {

    enum ActionName {FIRE,RIGHT,LEFT, AHEAD, BACK};
    // want an instance variable "action"
    ActionName action;
    double QValue = 0;

    public Action (String name) {

        switch (name) {
            case "fire":
                action = ActionName.FIRE;
                break;
            case "right":
                action = ActionName.RIGHT;
                break;
            case "left":
                action = ActionName.LEFT;
                break;
            case "ahead":
                action = ActionName.AHEAD;
                break;
            case "back":
                action = ActionName.BACK;
                break;
        }
    }
    public static void main(String[] args) {
        Action action = new Action("right");
        System.out.println(action.action);
    }


}
```

Listing 35: representation/Action.java

```java
package representation;

import java.io.Serializable;

public class Concatenation implements IRepresentable, Serializable {
    private IRepresentable first;
    private IRepresentable second;

    public Concatenation(IRepresentable state, IRepresentable action) {
        this.first = state;
        this.second = action;
    }
    @Override
```

```
14      public double[] toVector() {
15          double[] stateVector = first.toVector();
16          double[] actionVector = second.toVector();
17          double[] result = new double[stateVector.length + actionVector.
                length];
18          System.arraycopy(stateVector, 0, result, 0, stateVector.length);
19          System.arraycopy(actionVector, 0, result, stateVector.length,
                actionVector.length);
20          return result;
21      }
22
23      @Override
24      public int hashCode() {
25          return first.hashCode() + second.hashCode();
26      }
27
28      @Override
29      public boolean equals(Object obj) {
30          if (!(obj instanceof IRepresentable)) return false;
31          var testRepr = ((IRepresentable) obj).toVector();
32          double[] result = toVector();
33          if (testRepr.length != result.length) return false;
34          for (int i = 0; i < result.length; i++) {
35              if (result[i] != testRepr[i]) return false;
36          }
37          return true;
38      }
39
40      @Override
41      public String toString() {
42          return "Concatenation{" +
43                  "first=" + first +
44                  ", second=" + second +
45                  '}';
46      }
47  }
```

Listing 36: representation/Concatenation.java

```
1   package representation;
2
3   public class ConcatenationRepresentation implements
        IStateActionRepresentation {
4
5
6
7       @Override
8       public IRepresentable represent(IState state, IAction action) {
9           return new Concatenation(state, action);
10      }
11  }
```

Listing 37: representation/ConcatenationRepresentation.java

```
1   package representation;
2
3   import java.io.Serializable;
4   import java.util.Objects;
```

```java

public class Coordinates implements IState, Serializable {
    private int x;
    private int y;
    private int heading;

    public Coordinates(int x, int y, int heading) {
        setX(x);
        setY(y);
        setHeading(heading);
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void setHeading(int bearing) {
        this.heading = bearing;
    }

    @Override
    public IState clone() {
        return new Coordinates(this.x, this.y, this.heading);
    }

    public int getX() {
        return this.x;
    }

    @Override
    public double[] toVector() {
        return new double[] {x, y, heading};
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Coordinates that = (Coordinates) o;
        return x == that.x && y == that.y && heading == that.heading;
    }

    @Override
    public int hashCode() {
        return Objects.hash(x, y, heading);
    }

    public int getY() {
        return y;
    }

    @Override
    public String toString() {
        return "Coordinates{" +
```

```
                "x=" + x +
                ", y=" + y +
                ", heading=" + heading +
                '}';
    }
}
```

Listing 38: representation/Coordinates.java

```
package representation;

import robocode.Event;
import robocode.ScannedRobotEvent;
import robocode.StatusEvent;

public class CoordinatesRepresentation implements IStateRepresentation {
    @Override
    public IState represent(IState state, Event event) {
        if (state == null) {
            state = new Coordinates(0, 0, 0);
        }
        Coordinates coordinates = (Coordinates) state.clone();
        if (event instanceof StatusEvent) {
            StatusEvent statusEvent = (StatusEvent) event;
            coordinates.setX((int) (statusEvent.getStatus().getX() / 100));
            coordinates.setY((int) (statusEvent.getStatus().getY() / 100));
            coordinates.setHeading((int) ((statusEvent.getStatus().
                getHeading() + 45) / 90));
        }
        return coordinates;
    }
}
```

Listing 39: representation/CoordinatesRepresentation.java

```
package representation;

public interface IAction extends IRepresentable {

}
```

Listing 40: representation/IAction.java

```
package representation;

import robocode.Robot;

public interface IActionRepresentation extends IRepresentation {
    void takeAction(Robot robot, IAction action);

    IAction[] getActions();
}
```

Listing 41: representation/IActionRepresentation.java

```
package representation;

import java.io.Serializable;
```

```
4
5  public interface IRepresentable extends Serializable {
6      double[] toVector();
7  }
```

Listing 42: representation/IRepresentable.java

```
1  package representation;
2
3  import robocode.Robot;
4  import robocode.Event;
5
6  public interface IRepresentation {
7  }
```

Listing 43: representation/IRepresentation.java

```
1  package representation;
2
3  public interface IStateActionRepresentation {
4      IRepresentable represent(IState state, IAction action);
5  }
```

Listing 44: representation/IStateActionRepresentation.java

```
1  package representation;
2
3  public interface IState extends IRepresentable {
4      public IState clone();
5  }
```

Listing 45: representation/IState.java

```
1  package representation;
2
3  import robocode.Event;
4
5  public interface IStateRepresentation extends IRepresentation {
6      /**
7       * Evolves the robot state given the last state and the event
8       * @param state the previous state of the robot
9       * @param event the robot event containing changes to the state
10      * @return returns a new state expressing the changed state
11      */
12     IState represent(IState state, Event event);
13 }
```

Listing 46: representation/IStateRepresentation.java

```
1  package representation;
2
3  import java.io.Serializable;
4  import java.util.Objects;
5
6  public class Move implements IAction, Serializable {
7      @Override
8      public double[] toVector() {
9          double value = 0;
```

```
        switch (actionType) {
            case AHEAD:
                value = 1;
                break;
            case TURN_LEFT:
                value = 2;
                break;
            case TURN_RIGHT:
                value = 3;
                break;
            default:
                assert false;
        }
        return new double[] { value };
    }

    @Override
    public String toString() {
        return "Move{" +
                "actionType=" + actionType +
                '}';
    }

    public enum ActionType {
        TURN_RIGHT,
        TURN_LEFT,
        AHEAD,
    }

    ActionType actionType;

    public Move(ActionType actionType) {
        this.actionType = actionType;
    }

    public ActionType getActionType() {return actionType;}

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Move move = (Move) o;
        return actionType == move.actionType;
    }

    @Override
    public int hashCode() {
        return Objects.hash(actionType);
    }
}
```

Listing 47: representation/Move.java

```
package representation;

import robocode.Robot;

public class MoveRepresentation implements IActionRepresentation {
```

```
    @Override
    public void takeAction(Robot qLearningRobot, IAction action) {
        if (action == null) return;
        if (!(action instanceof Move)) {
            throw new IllegalArgumentException("Move representation can
                only take move actions.");
        }
        Move move = (Move) action;
        System.out.println("CASTED");
        if (move.getActionType() == Move.ActionType.TURN_LEFT) {
            qLearningRobot.turnLeft(90);
        } else if (move.getActionType() == Move.ActionType.TURN_RIGHT) {
            qLearningRobot.turnRight(90);
        } else if (move.getActionType() == Move.ActionType.AHEAD) {
            qLearningRobot.ahead(100);
        }
    }

    @Override
    public IAction[] getActions() {
        return new IAction[] {
            new Move(Move.ActionType.AHEAD),
            new Move(Move.ActionType.TURN_LEFT),
            new Move(Move.ActionType.TURN_RIGHT),
        };
    }
}
```

Listing 48: representation/MoveRepresentation.java

```
package representation;

public class Representation {

}
```

Listing 49: representation/Representation.java

```
package representation;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class State {

    // The states

    // the relative distance to the enemy (<200, <300, >=300)
    // our energy (<30, >=30, >100)
    // the enemy's energy (<30, >=30, >100)
    // x, y position of our own (step by 100)

    // The actions

    // fire(1)
    // turn right (90)
    // turn left (90)
```

```
21      // go ahead (100)
22      // go back (100)
23      // do nothing
24
25      private int distance;
26      private int energy;
27      private int enemyEnergy;
28      private int x;
29      private int y;
30      private int enemyBearing;
31
32      List<Action> actions = new ArrayList<Action>();
33
34      public void add(Action a) {
35          actions.add(a);
36      }
37
38      public void addAll() {
39          actions.add(new Action("fire"));
40          actions.add(new Action("right"));
41          actions.add(new Action("left"));
42          actions.add(new Action("ahead"));
43          actions.add(new Action("back"));
44      }
45
46      @Override
47      public boolean equals(Object o) {
48          if (this == o) return true;
49          if (o == null || getClass() != o.getClass()) return false;
50          State state = (State) o;
51          return distance == state.distance && energy == state.energy &&
                enemyEnergy == state.enemyEnergy && x == state.x;
52      }
53
54      @Override
55      public int hashCode() {
56          return Objects.hash(distance, energy, enemyEnergy, x);
57      }
58
59
60
61
62  }
```

Listing 50: representation/State.java

```
1   package representation;
2   import robocode.*;
3
4   public class StateRep implements IStateRepresentation {
5       public StateRep()  {}
6       @Override
7       // represent method will be called under two circumstances: either from
            onStatus or from onScannedRobot.
8       // the event passed in can be of either StatusEvent or
           ScannedRobotEvent
9       public IState represent(IState state, Event event) {
10          //passed states are the last states, all null at first turn
```

```
11          if (state == null) {
12              state = new States(0, 0, 0, 0, 0 , 0, 0);
13          }
14          States states = (States) state.clone();//cast State to States
15
16          if (event instanceof ScannedRobotEvent) {
17              ScannedRobotEvent scannedEvent = (ScannedRobotEvent) event;
18              states.setDistance((int) (scannedEvent.getDistance()));
19              states.setEnemyEnergy((int) scannedEvent.getEnergy());
20              states.setBearing((int)scannedEvent.getBearing());
21          }
22          if (event instanceof StatusEvent) {
23              StatusEvent statusEvent = (StatusEvent) event;
24              states.setX((int) statusEvent.getStatus().getX());
25              states.setY((int) statusEvent.getStatus().getY());
26              states.setHeading((int) statusEvent.getStatus().getHeading());
27              states.setMyEnergy((int) statusEvent.getStatus().getEnergy());
28          }
29          if (event instanceof WinEvent) {
30              states.setEnemyEnergy(0);
31          }
32          if (event instanceof DeathEvent) {
33              states.setMyEnergy(0);
34          }
35          return states;
36      }
37
38  }
```

Listing 51: representation/StateRep.java

```
1   package representation;
2
3   import java.util.Arrays;
4   import java.util.Objects;
5
6   public class States implements IState{
7
8       private int distance;
9       private int x;
10      private int y;
11      private int heading;
12      private int bearing;
13
14      public void setBearing(int bearing) {
15          this.bearing = bearing;
16      }
17
18      public int getBearing() {
19          return bearing;
20      }
21
22      public enum energy {LOW, MEDIUM, HIGH};
23      private int myEnergy;
24      private int enemyEnergy;
25
26      public States(int distance, int x, int y, int heading, int myEnergy,
            int enemyEnergy, int bearing) {
```

```
27          setDistance(distance);
28          setX(x);
29          setY(y);
30          setHeading(heading);
31          setMyEnergy(myEnergy);
32          setEnemyEnergy(enemyEnergy);
33          setBearing(bearing);
34      }
35
36      public void setDistance(int distance) {
37          this.distance = distance;
38      }
39      public void setX(int x) {
40          this.x = x;
41      }
42      public void setY(int y) {this.y = y;}
43      public void setHeading(int heading) {
44          this.heading = heading;
45      }
46      public void setMyEnergy(int myEnergy) {
47          this.myEnergy = myEnergy;
48      }
49      public void setEnemyEnergy(int enemyEnergy) {
50          this.enemyEnergy = enemyEnergy;
51      }
52
53      @Override
54      public IState clone() {
55          return new States(this.distance, this.x, this.y, this.heading, this
                .myEnergy, this.enemyEnergy, this.bearing);
56      }
57
58      @Override
59      public double[] toVector() {
60          return new double[]{
61  //                  this.x / 200,
62  //                  this.y / 200,
63  //                  (this.heading + 45) / 90,
64  //                  (this.bearing + 45) / 90,
65  //                  this.myEnergy / 40,
66  //                  this.enemyEnergy / 40,
67                    this.distance / 200,
68          };
69      }
70
71      public int getDistance() {
72          return distance;
73      }
74
75      public int getX() {
76          return x;
77      }
78
79      public int getY() {
80          return y;
81      }
82
83      public int getHeading() {
```

```
84          return heading;
85      }
86
87      public int getMyEnergy() {
88          return myEnergy;
89      }
90
91      public int getEnemyEnergy() {
92          return enemyEnergy;
93      }
94
95      @Override
96      public boolean equals(Object o) {
97          if (this == o) return true;
98          if (o == null || getClass() != o.getClass()) return false;
99          States states = (States) o;
100         double[] mine = toVector();
101         double[] theirs = states.toVector();
102         for (int i = 0; i < mine.length; i++) {
103             if (mine[i] != theirs[i]) {
104                 return false;
105             }
106         }
107         return true;
108     }
109
110     @Override
111     public int hashCode() {
112         return Arrays.hashCode(toVector());
113     }
114
115     @Override
116     public String toString() {
117         return "States{" +
118                 "distance=" + distance +
119                 ", x=" + x +
120                 ", y=" + y +
121                 ", heading=" + heading +
122                 ", bearing=" + bearing +
123                 ", myEnergy=" + myEnergy +
124                 ", enemyEnergy=" + enemyEnergy +
125                 '}';
126     }
127 }
```

Listing 52: representation/States.java

```
1  package representation;
2
3  import java.io.Serializable;
4  import java.util.Objects;
5
6  public class TNinetyAction implements IAction, Serializable {
7      @Override
8      public double[] toVector() {
9          double value = 0;
10         switch (actionType) {
11             case AHEAD:
```

```
                value = 1;
                break;
            case TURN_LEFT:
                value = 2;
                break;
            case TURN_RIGHT:
                value = 3;
                break;
            case FIRE:
                value = 4;
                break;
            default:
                assert false;
        }
        return new double[] { value };
    }

    @Override
    public String toString() {
        return "Move{" +
                "actionType=" + actionType +
                '}';
    }

    public enum ActionType {
        TURN_RIGHT,
        TURN_LEFT,
        AHEAD,
        FIRE,
        RANDOMLY_MOVE,
    }

    ActionType actionType;

    public TNinetyAction(ActionType actionType) {
        this.actionType = actionType;
    }

    public ActionType getActionType() {return actionType;}

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        TNinetyAction that = (TNinetyAction) o;
        return actionType == that.actionType;
    }

    @Override
    public int hashCode() {
        return Objects.hash(actionType);
    }
}
```

Listing 53: representation/TNinetyAction.java

```
package representation;

```

```java
import robocode.Robot;

import java.util.Random;

public class TNinetyActionRepresentation implements IActionRepresentation {
    @Override
    public void takeAction(Robot qLearningRobot, IAction action) {
        if (action == null) return;
        if (!(action instanceof TNinetyAction)) {
            throw new IllegalArgumentException("TNinety representation can only take TNinety actions.");
        }
        TNinetyAction move = (TNinetyAction) action;
        if (move.getActionType() == TNinetyAction.ActionType.TURN_LEFT) {
            qLearningRobot.turnLeft(90);
        } else if (move.getActionType() == TNinetyAction.ActionType.TURN_RIGHT) {
            qLearningRobot.turnRight(90);
        } else if (move.getActionType() == TNinetyAction.ActionType.AHEAD) {
            qLearningRobot.ahead(100);
        } else if (move.getActionType() == TNinetyAction.ActionType.FIRE) {
            qLearningRobot.fire(18);
        } else if (move.getActionType() == TNinetyAction.ActionType.RANDOMLY_MOVE) {
            MoveRepresentation moveRepresentation = new MoveRepresentation();
            System.out.println("TAKING RANDOM");
            moveRepresentation.takeAction(qLearningRobot, moveRepresentation.getActions()[new Random().nextInt(3)]);
        }
    }

    @Override
    public IAction[] getActions() {
        return new IAction[] {
            new TNinetyAction(TNinetyAction.ActionType.AHEAD),
            new TNinetyAction(TNinetyAction.ActionType.TURN_LEFT),
//            new TNinetyAction(TNinetyAction.ActionType.TURN_RIGHT),
            new TNinetyAction(TNinetyAction.ActionType.FIRE),
//            new TNinetyAction(TNinetyAction.ActionType.RANDOMLY_MOVE),
        };
    }
}
```

Listing 54: representation/TNinetyActionRepresentation.java

```java
package rl;

import fa.IFunctionApproximation;
import fa.LUT;
import policy.IPolicy;
import representation.*;

public interface ILearning {
    IAction takeStep(IState lastState, IAction lastAction, IState currentState);

```

```
    IStateRepresentation getStateRepresentation();
    IActionRepresentation getActionRepresentation();

    IPolicy getPolicy();

    IFunctionApproximation getFunctionApproximation();
}
```

Listing 55: rl/ILearning.java

```
package rl;

import fa.IFunctionApproximation;
import org.jetbrains.annotations.NotNull;
import policy.IPolicy;
import representation.*;

import java.util.ArrayList;
import java.util.Random;

public class QLearning implements ILearning {

    private double epsilon;
    private final double alpha;
    private final double gamma;
    private final Random random;
    private IStateRepresentation stateRepresentation;
    private IActionRepresentation actionRepresentation;
    private IPolicy policy;
    private IFunctionApproximation functionApproximation;
    private IStateActionRepresentation stateActionRepresentation;
    private int depth;
    private boolean onlineLearning = false;
    private ArrayList<IRepresentable> history = new ArrayList<>();
    private ConcatenationRepresentation concatenation = new
        ConcatenationRepresentation();

    public QLearning(IStateRepresentation stateRepresentation,
                     IActionRepresentation actionRepresentation,
                     IStateActionRepresentation stateActionRepresentation,
                     IPolicy policy, IFunctionApproximation
                         functionApproximation, double epsilon, double alpha
                         , double gamma) {
        this.stateActionRepresentation = stateActionRepresentation;
        this.stateRepresentation = stateRepresentation;
        this.actionRepresentation = actionRepresentation;
        this.policy = policy;
        this.epsilon = epsilon;
        this.alpha = alpha;
        this.gamma = gamma;
        this.random = new Random();
        this.functionApproximation = functionApproximation;
        this.depth = 1;
        this.onlineLearning = false;
    }


    public QLearning(IStateRepresentation stateRepresentation,
```

```
46                      IActionRepresentation actionRepresentation,
47                      IStateActionRepresentation stateActionRepresentation,
48                      IPolicy policy, IFunctionApproximation
                            functionApproximation,
49                      double epsilon, double alpha, double gamma, int depth,
                            boolean onlineLearning) {        this.epsilon =
                            epsilon;
50          this.alpha = alpha;
51          this.gamma = gamma;
52          this.stateRepresentation = stateRepresentation;
53          this.actionRepresentation = actionRepresentation;
54          this.policy = policy;
55          this.functionApproximation = functionApproximation;
56          this.stateActionRepresentation = stateActionRepresentation;
57          this.depth = depth;
58          this.random = new Random();
59          this.onlineLearning = onlineLearning;
60      }
61
62      @Override
63      public IAction takeStep(IState lastState, IAction lastAction, IState
            currentState) {
64          if (lastState == null || lastAction == null || currentState == null
                ) {
65              return explore();
66          }
67          System.out.println("Current state " + currentState);
68          IRepresentable oldSA = stateActionRepresentation.represent(
69                  lastState,
70                  lastAction
71          );
72          history.add(oldSA);
73          while (history.size() > depth) {
74              history.remove(0);
75          }
76          if (history.size() < depth) return explore();
77          IAction bestAction = exploit(currentState);
78          IAction toTakeAction;
79          if (this.random.nextDouble() < this.epsilon) {
80              IAction action = explore();
81              logAction(currentState, action, "explored");
82              toTakeAction = action;
83          } else {
84              logAction(currentState, bestAction, "exploited");
85              toTakeAction = bestAction;
86          }
87          if (!onlineLearning) {
88              toTakeAction = bestAction;
89          }
90
91          for (int i = 0; i < history.size() - 1; i++) {
92              oldSA = new Concatenation(oldSA, history.get(i));
93          }
94          IRepresentable currentAction = stateActionRepresentation.represent(
                currentState, toTakeAction);
95          for (int i = 1; i < history.size(); i++) {
96              currentAction = new Concatenation(currentAction, history.get(i)
                    );
```

```
97          }
98          double oldQ = functionApproximation.eval(oldSA)[0];
99          double r = policy.getReward(currentState, lastState); // why would
                evaluate Rewards for last state?
100
101
102         double currentQ = functionApproximation.eval(currentAction)[0];
103         System.out.println("train " + oldQ + " = " + oldQ + " + " + alpha +
                " (" + r + " + " + gamma + " * " + currentQ  + " - " + oldQ + "
                )");
104         System.out.println("train " + oldSA + " " + toTakeAction);
105         functionApproximation.train(
106                 oldSA,
107                 new double[]{
108                         oldQ + alpha * (r + gamma * currentQ - oldQ)
109                 }
110         );
111         return toTakeAction;
112     }
113
114     private IAction exploit(IState currentState) {
115         IAction bestAction = actionRepresentation.getActions()[0];
116         IRepresentable stateAction = stateActionRepresentation.represent(
                currentState, bestAction);
117         for (int i = 1; i < history.size(); i++) {
118             stateAction = new Concatenation(stateAction, history.get(i));
119         }
120         double bestQ = functionApproximation.eval(stateAction)[0];
121
122         for (IAction action: actionRepresentation.getActions()) {
123             stateAction = stateActionRepresentation.represent(
124                     currentState, action
125             );
126             for (int i = 1; i < history.size(); i++) {
127                 stateAction = new Concatenation(stateAction, history.get(i)
                        );
128             }
129             double q = functionApproximation.eval(
130                     stateAction)[0];
131             System.out.print("    " + q + " " + action);
132             System.out.println();
133             if (q > bestQ) {
134                 bestAction = action;
135                 bestQ = q;
136             }
137         }
138         return bestAction;
139     }
140
141     private void logAction(IState currentState, IAction bestAction, String
            hint) {
142         if (bestAction == null) return;
143         System.out.print(bestAction);
144         System.out.print(" " + hint + " ");
145         System.out.println(functionApproximation.eval(
146                 stateActionRepresentation.represent(
147                         currentState, bestAction
148                 ))[0]);
```

```
149     }
150
151     private IAction explore() {
152         IAction[] actions = actionRepresentation.getActions();
153         IAction action = actions[getRandom().nextInt(actions.length)];
154         return action;
155     }
156
157     @NotNull
158     private Random getRandom() {
159         return this.random;
160     }
161
162     @Override
163     public IStateRepresentation getStateRepresentation() {
164         return stateRepresentation;
165     }
166
167     @Override
168     public IActionRepresentation getActionRepresentation() {
169         return actionRepresentation;
170     }
171
172     @Override
173     public IPolicy getPolicy() {
174         return policy;
175     }
176
177     @Override
178     public IFunctionApproximation getFunctionApproximation() {
179         return functionApproximation;
180     }
181
182     public void setEpsilon(double epsilon) {
183         this.epsilon = epsilon;
184     }
185
186     public double getEpsilon() {
187         return this.epsilon;
188     }
189 }
```

Listing 56: rl/QLearning.java

```
1  package rl;
2
3  import fa.IFunctionApproximation;
4  import org.jetbrains.annotations.NotNull;
5  import policy.IPolicy;
6  import representation.*;
7
8  import java.util.Random;
9
10 public class SARSALearning implements ILearning {
11
12     private double epsilon;
13     private final double alpha;
14     private final double gamma;
```

```
15      private final Random random;
16      private IStateRepresentation stateRepresentation;
17      private IActionRepresentation actionRepresentation;
18      private IPolicy policy;
19      private IFunctionApproximation functionApproximation;
20      private IStateActionRepresentation stateActionRepresentation;
21
22      public SARSALearning(IStateRepresentation stateRepresentation,
23                           IActionRepresentation actionRepresentation,
24                           IStateActionRepresentation
                                 stateActionRepresentation,
25                           IPolicy policy, IFunctionApproximation
                                 functionApproximation, double epsilon, double
                                 alpha, double gamma) {
26          this.stateActionRepresentation = stateActionRepresentation;
27          this.stateRepresentation = stateRepresentation;
28          this.actionRepresentation = actionRepresentation;
29          this.policy = policy;
30          this.epsilon = epsilon;
31          this.alpha = alpha;
32          this.gamma = gamma;
33          this.random = new Random();
34          this.functionApproximation = functionApproximation;
35      }
36
37      @Override
38      public IAction takeStep(IState lastState, IAction lastAction, IState
            currentState) {
39          if (lastState == null || lastAction == null || currentState == null
                ) {
40              return explore();}
41          System.out.println("Current state " + currentState);
42          IRepresentable oldSA = stateActionRepresentation.represent(
43                  lastState,
44                  lastAction);
45          double oldQ = functionApproximation.eval(oldSA)[0];
46          double r = policy.getReward(currentState, lastState); // why would
                evaluate Rewards for last state?
47          double newQ;
48
49          if (this.random.nextDouble() < this.epsilon) {
50              IAction action = explore();
51              IRepresentable exploreSA = stateActionRepresentation.represent(
52                      currentState,
53                      action);
54              newQ = functionApproximation.eval(exploreSA)[0];
55              System.out.println("train " + oldQ + " = " + oldQ + " + " +
                    alpha + " (" + r + " + " + gamma + " * " + newQ  + " - " +
                    oldQ + ")");
56              System.out.println("train " + oldSA + " " + action);
57              functionApproximation.train(
58                      oldSA,
59                      new double[]{
60                              oldQ + alpha * (r + gamma * newQ - oldQ)
61                      });
62              logAction(currentState, action, "explored");
63              return action;
64          } else {
```

```
65              IAction bestAction = exploit(currentState);
66              IRepresentable currentBest = stateActionRepresentation.
                    represent(
67                      currentState,
68                      bestAction);
69              newQ = functionApproximation.eval(currentBest)[0];
70              System.out.println("train " + oldQ + " = " + oldQ + " + " +
                    alpha + " (" + r + " + " + gamma + " * " + newQ  + " - " +
                    oldQ + ")");
71              System.out.println("train " + oldSA + " " + bestAction);
72              functionApproximation.train(
73                      oldSA,
74                      new double[]{
75                              oldQ + alpha * (r + gamma * newQ - oldQ)
76                      });
77              logAction(currentState, bestAction, "exploited");
78              return bestAction;
79          }
80      }
81
82      private IAction exploit(IState currentState) {
83          double bestQ = 0;
84          IAction bestAction = actionRepresentation.getActions()[0];
85          for (IAction action: actionRepresentation.getActions()) {
86              double q = functionApproximation.eval(
87                      stateActionRepresentation.represent(
88                              currentState, action
89                      ))[0];
90              System.out.print("    " + q + " " + action);
91              System.out.println();
92              if (q > bestQ) {
93                  bestAction = action;
94                  bestQ = q;
95              }
96          }
97          return bestAction;
98      }
99
100     private void logAction(IState currentState, IAction bestAction, String
            hint) {
101         if (bestAction == null) return;
102         System.out.print(bestAction);
103         System.out.print(" " + hint + " ");
104         System.out.println(functionApproximation.eval(
105                 stateActionRepresentation.represent(
106                         currentState, bestAction
107                 ))[0]);
108     }
109
110     private IAction explore() {
111         IAction[] actions = actionRepresentation.getActions();
112         IAction action = actions[getRandom().nextInt(actions.length)];
113         return action;
114     }
115
116     @NotNull
117     private Random getRandom() {
118         return this.random;
```

```
119     }
120
121     @Override
122     public IStateRepresentation getStateRepresentation() {
123         return stateRepresentation;
124     }
125
126     @Override
127     public IActionRepresentation getActionRepresentation() {
128         return actionRepresentation;
129     }
130
131     @Override
132     public IPolicy getPolicy() {
133         return policy;
134     }
135
136     @Override
137     public IFunctionApproximation getFunctionApproximation() {
138         return functionApproximation;
139     }
140
141     public void setEpsilon(double epsilon) {
142         this.epsilon = epsilon;
143     }
144
145     public double getEpsilon() {
146         return this.epsilon;
147     }
148 }
```

Listing 57: rl/SARSALearning.java

```
1  /*
2   * Copyright (c) 2001-2021 Mathew A. Nelson and Robocode contributors
3   * All rights reserved. This program and the accompanying materials
4   * are made available under the terms of the Eclipse Public License v1.0
5   * which accompanies this distribution, and is available at
6   * https://robocode.sourceforge.io/license/epl-v10.html
7   */
8  package robot;
9
10
11 import robocode.DeathEvent;
12 import robocode.Robot;
13 import robocode.ScannedRobotEvent;
14 import static robocode.util.Utils.normalRelativeAngleDegrees;
15
16 import java.awt.*;
17
18
19 /**
20  * Corners - a sample robot by Mathew Nelson.
21  * <p>
22  * This robot moves to a corner, then swings the gun back and forth.
23  * If it dies, it tries a new corner in the next round.
24  *
25  * @author Mathew A. Nelson (original)
```

```
 * @author Flemming N. Larsen (contributor)
 */
public class Corners extends Robot {
    int others; // Number of other robots in the game
    static int corner = 0; // Which corner we are currently using
    // static so that it keeps it between rounds.
    boolean stopWhenSeeRobot = false; // See goCorner()

    /**
     * run:  Corners' main run function.
     */
    public void run() {
        // Set colors
        setBodyColor(Color.red);
        setGunColor(Color.black);
        setRadarColor(Color.yellow);
        setBulletColor(Color.green);
        setScanColor(Color.green);

        // Save # of other bots
        others = getOthers();

        // Move to a corner
        goCorner();

        // Initialize gun turn speed to 3
        int gunIncrement = 3;

        // Spin gun back and forth
        while (true) {
            for (int i = 0; i < 30; i++) {
                turnGunLeft(gunIncrement);
            }
            gunIncrement *= -1;
        }
    }

    /**
     * goCorner:  A very inefficient way to get to a corner.  Can you do
         better?
     */
    public void goCorner() {
        // We don't want to stop when we're just turning...
        stopWhenSeeRobot = false;
        // turn to face the wall to the "right" of our desired corner.
        turnRight(normalRelativeAngleDegrees(corner - getHeading()));
        // Ok, now we don't want to crash into any robot in our way...
        stopWhenSeeRobot = true;
        // Move to that wall
        ahead(5000);
        // Turn to face the corner
        turnLeft(90);
        // Move to the corner
        ahead(5000);
        // Turn gun to starting point
        turnGunLeft(90);
    }

```

```
83      /**
84       * onScannedRobot:  Stop and fire!
85       */
86      public void onScannedRobot(ScannedRobotEvent e) {
87          // Should we stop, or just fire?
88          if (stopWhenSeeRobot) {
89              // Stop everything!  You can safely call stop multiple times.
90              stop();
91              // Call our custom firing method
92              smartFire(e.getDistance());
93              // Look for another robot.
94              // NOTE:  If you call scan() inside onScannedRobot, and it sees
                    a robot,
95              // the game will interrupt the event handler and start it over
96              scan();
97              // We won't get here if we saw another robot.
98              // Okay, we didn't see another robot... start moving or turning
                    again.
99              resume();
100         } else {
101             smartFire(e.getDistance());
102         }
103     }
104
105     /**
106      * smartFire:  Custom fire method that determines firepower based on
            distance.
107      *
108      * @param robotDistance the distance to the robot to fire at
109      */
110     public void smartFire(double robotDistance) {
111         if (robotDistance > 200 || getEnergy() < 15) {
112             fire(1);
113         } else if (robotDistance > 50) {
114             fire(2);
115         } else {
116             fire(3);
117         }
118     }
119
120     /**
121      * onDeath:  We died.  Decide whether to try a different corner next
            game.
122      */
123     public void onDeath(DeathEvent e) {
124         // Well, others should never be 0, but better safe than sorry.
125         if (others == 0) {
126             return;
127         }
128
129         // If 75% of the robots are still alive when we die, we'll switch
                corners.
130         if ((others - getOthers()) / (double) others < .75) {
131             corner += 90;
132             if (corner == 270) {
133                 corner = -90;
134             }
135             out.println("I died and did poorly... switching corner to " +
```

```
                corner);
136         } else {
137             out.println("I died but did well.  I will still use corner " +
                corner);
138         }
139     }
140 }
```

Listing 58: robot/Corners.java

```
1  package robot;
2
3  import fa.IFunctionApproximation;
4  import fa.LUT;
5  import policy.EnergyReward;
6  import policy.IPolicy;
7  import representation.*;
8  import rl.ILearning;
9  import rl.QLearning;
10 import robocode.ScannedRobotEvent;
11
12 public class LUTTNinetyRobot05 extends QLearningRobot {
13     public LUTTNinetyRobot05() {
14         super(createLearning());
15     }
16
17
18     public static ILearning createLearning() {
19         IActionRepresentation actionRepresentation = new
               TNinetyActionRepresentation();
20         IStateRepresentation stateRepresentation = new StateRep();
21         IFunctionApproximation functionApproximation = new LUT("
               LUTTNinetyRobot.obj", false);
22         IPolicy policy = new EnergyReward();
23         return new QLearning(
24                 stateRepresentation,
25                 actionRepresentation,
26                 new ConcatenationRepresentation(),
27                 policy,
28                 functionApproximation,
29                 0.5, 0.1, 0.9, 3, false);
30     }
31
32     @Override
33     public void onScannedRobot(ScannedRobotEvent event) {
34         super.onScannedRobot(event);
35         System.out.println("HELLLLLOOOOOOO");
36     }
37 }
```

Listing 59: robot/LUTTNinetyRobot05.java

```
1  package robot;
2
3  import fa.IFunctionApproximation;
4  import fa.LUT;
5  import policy.EnergyReward;
6  import policy.IPolicy;
```

```java
import representation.*;
import rl.ILearning;
import rl.QLearning;
import robocode.ScannedRobotEvent;

public class LUTTNinetyRobot0 extends QLearningRobot {
    public LUTTNinetyRobot0() {
        super(createLearning());
    }


    public static ILearning createLearning() {
        IActionRepresentation actionRepresentation = new
            TNinetyActionRepresentation();
        IStateRepresentation stateRepresentation = new StateRep();
        IFunctionApproximation functionApproximation = new LUT("
            LUTTNinetyRobot.obj", false);
        IPolicy policy = new EnergyReward();
        return new QLearning(
                stateRepresentation,
                actionRepresentation,
                new ConcatenationRepresentation(),
                policy,
                functionApproximation,
                0.0, 0.1, 0.9, 3, false);
    }

    @Override
    public void onScannedRobot(ScannedRobotEvent event) {
        super.onScannedRobot(event);
        System.out.println("HELLLLLOOOOOOO");
    }
}
```

Listing 60: robot/LUTTNinetyRobot0.java

```java
package robot;

import fa.IFunctionApproximation;
import fa.LUT;
import policy.EnergyReward;
import policy.IPolicy;
import representation.*;
import rl.ILearning;
import rl.QLearning;
import robocode.ScannedRobotEvent;

public class LUTTNinetyRobotConfident extends QLearningRobot {
    public LUTTNinetyRobotConfident() {
        super(createLearning());
    }


    public static ILearning createLearning() {
        IActionRepresentation actionRepresentation = new
            TNinetyActionRepresentation();
        IStateRepresentation stateRepresentation = new StateRep();
        IFunctionApproximation functionApproximation = new LUT("
```

```
            LUTTNinetyRobot.obj", true);
        IPolicy policy = new EnergyReward();
        return new QLearning(
                stateRepresentation,
                actionRepresentation,
                new ConcatenationRepresentation(),
                policy,
                functionApproximation,
                0.05, 0.1, 0.8, 3, false);
    }

    @Override
    public void onScannedRobot(ScannedRobotEvent event) {
        super.onScannedRobot(event);
        System.out.println("HELLLLLOOOOOOO");
    }
}
```

Listing 61: robot/LUTTNinetyRobotConfident.java

```
package robot;

import fa.IFunctionApproximation;
import fa.LUT;
import policy.EnergyReward;
import policy.GoTopRight;
import policy.IPolicy;
import representation.*;
import rl.ILearning;
import rl.QLearning;
import robocode.ScannedRobotEvent;

public class LUTTNinetyRobot extends QLearningRobot {
    public LUTTNinetyRobot() {
        super(createLearning());
    }


    public static ILearning createLearning() {
        IActionRepresentation actionRepresentation = new
            TNinetyActionRepresentation();
        IStateRepresentation stateRepresentation = new StateRep();
        IFunctionApproximation functionApproximation = new LUT("
            LUTTNinetyRobot.obj", false);
        IPolicy policy = new EnergyReward();
        return new QLearning(
                stateRepresentation,
                actionRepresentation,
                new ConcatenationRepresentation(),
                policy,
                functionApproximation,
                0.8, 0.1, 0.9, 3, false);
    }

    @Override
    public void onScannedRobot(ScannedRobotEvent event) {
        super.onScannedRobot(event);
        System.out.println("HELLLLLOOOOOOO");
```

```
    }
}
```

Listing 62: robot/LUTTNinetyRobot.java

```
package robot;

import fa.IFunctionApproximation;
import fa.LUT;
import policy.EnergyReward;
import policy.IPolicy;
import representation.*;
import rl.ILearning;
import rl.QLearning;
import robocode.ScannedRobotEvent;

public class LUTTNinetyRobotOnline extends QLearningRobot {
    public LUTTNinetyRobotOnline() {
        super(createLearning());
    }


    public static ILearning createLearning() {
        IActionRepresentation actionRepresentation = new
            TNinetyActionRepresentation();
        IStateRepresentation stateRepresentation = new StateRep();
        IFunctionApproximation functionApproximation = new LUT("
            LUTTNinetyRobot.obj", false);
        IPolicy policy = new EnergyReward();
        return new QLearning(
                stateRepresentation,
                actionRepresentation,
                new ConcatenationRepresentation(),
                policy,
                functionApproximation,
                0.8, 0.1, 0.9, 3, true);
    }

    @Override
    public void onScannedRobot(ScannedRobotEvent event) {
        super.onScannedRobot(event);
        System.out.println("HELLLLLOOOOOOO");
    }
}
```

Listing 63: robot/LUTTNinetyRobotOnline.java

```
package robot;

import fa.IFunctionApproximation;
import fa.LUT;
import policy.EnergyReward;
import policy.EnergyRewardTerminal;
import policy.IPolicy;
import representation.*;
import rl.ILearning;
import rl.QLearning;
import robocode.ScannedRobotEvent;
```

```
12
13 public class LUTTNinetyRobotTerminal extends QLearningRobot {
14     public LUTTNinetyRobotTerminal() {
15         super(createLearning());
16     }
17
18
19     public static ILearning createLearning() {
20         IActionRepresentation actionRepresentation = new
               TNinetyActionRepresentation();
21         IStateRepresentation stateRepresentation = new StateRep();
22         IFunctionApproximation functionApproximation = new LUT("
               LUTTNinetyRobot.obj", false);
23         IPolicy policy = new EnergyRewardTerminal();
24         return new QLearning(
25                 stateRepresentation,
26                 actionRepresentation,
27                 new ConcatenationRepresentation(),
28                 policy,
29                 functionApproximation,
30                 0.8, 0.1, 0.9, 3, false);
31     }
32
33     @Override
34     public void onScannedRobot(ScannedRobotEvent event) {
35         super.onScannedRobot(event);
36         System.out.println("HELLLLLOOOOOOO");
37     }
38 }
```

Listing 64: robot/LUTTNinetyRobotTerminal.java

```
1 package robot;
2
3 /**
4  * DISCLAIMER: the code below has been auto-generated by Robocde
5  *    http://robocode.sourceforge.net/
6  */
7
8 import robocode.*;
9
10 // API help : http://robocode.sourceforge.net/docs/robocode/robocode/Robot.
       html
11
12 /**
13  * MyFirstRobot - a robot by (your name here)
14  */
15 public class MyFirstRobot extends AdvancedRobot
16 {
17     /**
18      * run: MyFirstRobot's default behavior
19      */
20     public void run() {
21         // Initialization of the robot should be put here
22
23         // After trying out your robot, try uncommenting the import at the
               top,
24         // and the next line:
```

```
25
26          // setColors(Color.red,Color.blue,Color.green); // body,gun,radar
27
28          // Robot main loop
29          while(true) {
30              // Replace the next 4 lines with any behavior you would like
31              ahead(100);
32              turnGunRight(360);
33              back(100);
34              turnGunRight(360);
35          }
36      }
37
38      /**
39       * onScannedRobot: What to do when you see another robot
40       */
41      public void onScannedRobot(ScannedRobotEvent e) {
42          // Replace the next line with any behavior you would like
43          fire(1);
44      }
45
46      /**
47       * onHitByBullet: What to do when you're hit by a bullet
48       */
49      public void onHitByBullet(HitByBulletEvent e) {
50          // Replace the next line with any behavior you would like
51          back(10);
52      }
53
54      /**
55       * onHitWall: What to do when you hit a wall
56       */
57      public void onHitWall(HitWallEvent e) {
58          // Replace the next line with any behavior you would like
59          back(20);
60      }
61  }
```

Listing 65: robot/MyFirstRobot.java

```
1   package robot;
2
3   import representation.IAction;
4   import representation.IState;
5   import rl.ILearning;
6   import robocode.*;
7
8   import java.io.IOException;
9
10  public class QLearningRobot extends Robot {
11      private ILearning learning;
12      private IState state;
13      private IState lastState;
14      private IAction lastAction;
15      private long lastTurn;
16      private int turn = 0;
17      private StatusEvent lastStatusEvent;
18
```

```
19      @Override
20      public void run() {
21          super.run();
22  //          setAdjustGunForRobotTurn(true);
23  //          setAdjustRadarForGunTurn(true);
24          while (true) {
25              // Replace the next 4 lines with any behavior you would like
26  //              ahead(100);
27  //              turnGunRight(360);
28  //              back(100);
29  //              turnRadarLeft(360);
30              turnGunRight(360);
31              turn ++;
32          }
33      }
34
35      public int getTurn() {
36          return turn;
37      }
38
39      public QLearningRobot(ILearning learning) {
40          this.learning = learning;
41          try {
42              this.learning.getFunctionApproximation().load();
43          } catch (IOException e) {
44              e.printStackTrace();
45          } catch (ClassNotFoundException e) {
46              e.printStackTrace();
47          }
48      }
49
50      public ILearning getLearning() {
51          return learning;
52      }
53
54
55      private IState getLastState() {
56          return lastState;
57      }
58
59      public IState getState() {
60          return this.state;
61      }
62
63      @Override
64      public void onScannedRobot(ScannedRobotEvent event) {
65          super.onScannedRobot(event);
66          processEvent(event);
67      }
68
69      private void processEvent(Event event) {
70          if (event instanceof ScannedRobotEvent && this.getTurn() > this.
                lastTurn || event instanceof WinEvent || event instanceof
                DeathEvent) {
71              IState newState = learning.getStateRepresentation().represent(
                    getState(), lastStatusEvent);
72              newState = learning.getStateRepresentation().represent(newState
                    , event);
```

```
            //set current state
            setState(newState);
            IAction action = learning.takeStep(getLastState(),
                getLastAction(), getState());
            //take new action
            this.lastTurn = this.getTurn();
            System.out.println("Turn " + this.lastTurn);
            takeAction(action);
        }
    }

    @Override
    public void onWin(WinEvent event) {
        super.onWin(event);
        processEvent(event);
    }

    @Override
    public void onDeath(DeathEvent event) {
        super.onDeath(event);
        processEvent(event);
    }

    private IAction getLastAction() {
        return lastAction;
    }

    private void takeAction(IAction action) {
        learning.getActionRepresentation().takeAction(this, action);
        if (action != null) lastAction = action;
    }

    public void setState(IState state) {
        this.lastState = this.state;
        this.state = state;
    }

    @Override
    public void onRoundEnded(RoundEndedEvent event) {
        try {
            learning.getFunctionApproximation().save();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onStatus(StatusEvent e) {
        lastStatusEvent = e;
    }
}
```

Listing 66: robot/QLearningRobot.java

```
package robot;

import robocode.Robot;
import robocode.StatusEvent;
```

```
5
6  public class TopLeftCornerRobot extends Robot {
7      @Override
8      public void run() {
9          super.run();
10         ahead(100);
11     }
12
13     @Override
14     public void onStatus(StatusEvent e) {
15         super.onStatus(e);
16         var yDistance = getBattleFieldHeight() - getY();
17         var xDistance = getX();
18         var angle = Math.atan(yDistance / xDistance);
19         angle = 270 + Math.toDegrees(angle);
20         System.out.printf("%f, %f, %f, %f %n", getY(), getX(), getHeading()
               , angle);
21         double absHeadingDiff = Math.abs(getHeading() - angle);
22         if (getHeading() < angle) {
23             if (absHeadingDiff > 180) {
24                 turnLeft(360 - absHeadingDiff);
25             } else {
26                 turnRight(absHeadingDiff);
27             }
28         } else {
29             if (absHeadingDiff > 180) {
30                 turnRight(360 - absHeadingDiff);
31             } else {
32                 turnLeft(absHeadingDiff);
33             }
34         }
35         if (absHeadingDiff < 0.01) {
36             ahead(100);
37         }
38     }
39 }
```

Listing 67: robot/TopLeftCornerRobot.java

```
1  /*
2   * Copyright (c) 2001-2021 Mathew A. Nelson and Robocode contributors
3   * All rights reserved. This program and the accompanying materials
4   * are made available under the terms of the Eclipse Public License v1.0
5   * which accompanies this distribution, and is available at
6   * https://robocode.sourceforge.io/license/epl-v10.html
7   */
8  package robot;
9
10
11 import robocode.HitRobotEvent;
12 import robocode.Robot;
13 import robocode.ScannedRobotEvent;
14 import robocode.WinEvent;
15 import static robocode.util.Utils.normalRelativeAngleDegrees;
16
17 import java.awt.*;
18
19
```

```
20 /**
21  * Tracker - a sample robot by Mathew Nelson.
22  * <p>
23  * Locks onto a robot, moves close, fires when close.
24  *
25  * @author Mathew A. Nelson (original)
26  * @author Flemming N. Larsen (contributor)
27  */
28 public class Tracker extends Robot {
29     int count = 0; // Keeps track of how long we've
30     // been searching for our target
31     double gunTurnAmt; // How much to turn our gun when searching
32     String trackName; // Name of the robot we're currently tracking
33 
34     /**
35      * run:  Tracker's main run function
36      */
37     public void run() {
38         // Set colors
39         setBodyColor(new Color(128, 128, 50));
40         setGunColor(new Color(50, 50, 20));
41         setRadarColor(new Color(200, 200, 70));
42         setScanColor(Color.white);
43         setBulletColor(Color.blue);
44 
45         // Prepare gun
46         trackName = null; // Initialize to not tracking anyone
47         setAdjustGunForRobotTurn(true); // Keep the gun still when we turn
48         gunTurnAmt = 10; // Initialize gunTurn to 10
49 
50         // Loop forever
51         while (true) {
52             // turn the Gun (looks for enemy)
53             turnGunRight(gunTurnAmt);
54             // Keep track of how long we've been looking
55             count++;
56             // If we've haven't seen our target for 2 turns, look left
57             if (count > 2) {
58                 gunTurnAmt = -10;
59             }
60             // If we still haven't seen our target for 5 turns, look right
61             if (count > 5) {
62                 gunTurnAmt = 10;
63             }
64             // If we *still* haven't seen our target after 10 turns, find
                   another target
65             if (count > 11) {
66                 trackName = null;
67             }
68         }
69     }
70 
71     /**
72      * onScannedRobot:  Here's the good stuff
73      */
74     public void onScannedRobot(ScannedRobotEvent e) {
75 
76         // If we have a target, and this isn't it, return immediately
```

```
77          // so we can get more ScannedRobotEvents.
78          if (trackName != null && !e.getName().equals(trackName)) {
79              return;
80          }
81
82          // If we don't have a target, well, now we do!
83          if (trackName == null) {
84              trackName = e.getName();
85              out.println("Tracking " + trackName);
86          }
87          // This is our target.  Reset count (see the run method)
88          count = 0;
89          // If our target is too far away, turn and move toward it.
90          if (e.getDistance() > 150) {
91              gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (
                    getHeading() - getRadarHeading()));
92
93              turnGunRight(gunTurnAmt); // Try changing these to
                    setTurnGunRight,
94              turnRight(e.getBearing()); // and see how much Tracker improves
                    ...
95              // (you'll have to make Tracker an AdvancedRobot)
96              ahead(e.getDistance() - 140);
97              return;
98          }
99
100         // Our target is close.
101         gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (
                getHeading() - getRadarHeading()));
102         turnGunRight(gunTurnAmt);
103         fire(3);
104
105         // Our target is too close!  Back up.
106         if (e.getDistance() < 100) {
107             if (e.getBearing() > -90 && e.getBearing() <= 90) {
108                 back(40);
109             } else {
110                 ahead(40);
111             }
112         }
113         scan();
114     }
115
116     /**
117      * onHitRobot:  Set him as our new target
118      */
119     public void onHitRobot(HitRobotEvent e) {
120         // Only print if he's not already our target.
121         if (trackName != null && !trackName.equals(e.getName())) {
122             out.println("Tracking " + e.getName() + " due to collision");
123         }
124         // Set the target
125         trackName = e.getName();
126         // Back up a bit.
127         // Note:  We won't get scan events while we're doing this!
128         // An AdvancedRobot might use setBack(); execute();
129         gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (
                getHeading() - getRadarHeading()));
```

```
130             turnGunRight(gunTurnAmt);
131             fire(3);
132             back(50);
133         }
134 
135         /**
136          * onWin:  Do a victory dance
137          */
138         public void onWin(WinEvent e) {
139             for (int i = 0; i < 50; i++) {
140                 turnRight(30);
141                 turnLeft(30);
142             }
143         }
144 }
```

Listing 68: robot/Tracker.java

```
1  package robot;
2 
3  import fa.IFunctionApproximation;
4  import fa.LUT;
5  import policy.EnergyReward;
6  import policy.GoTopRight;
7  import policy.IPolicy;
8  import representation.*;
9  import rl.ILearning;
10 import rl.QLearning;
11 
12 public class TrivialLUTRobotConfident extends QLearningRobot {
13     public TrivialLUTRobotConfident() {
14         super(createLearning());
15     }
16 
17 
18     public static ILearning createLearning() {
19         IActionRepresentation actionRepresentation = new MoveRepresentation
               ();
20         IStateRepresentation stateRepresentation = new
               CoordinatesRepresentation();
21         IFunctionApproximation functionApproximation = new LUT("
               TrivialLUTRobot.obj", true);
22         IPolicy policy = new GoTopRight();
23         return new QLearning(
24                 stateRepresentation,
25                 actionRepresentation,
26                 new ConcatenationRepresentation(),
27                 policy,
28                 functionApproximation,
29                 0.05, 0.1, 0.8);
30     }
31 }
```

Listing 69: robot/TrivialLUTRobotConfident.java

```
1  package robot;
2 
3  import fa.IFunctionApproximation;
```

```java
import fa.LUT;
import policy.EnergyReward;
import policy.GoTopRight;
import policy.IPolicy;
import representation.*;
import rl.ILearning;
import rl.QLearning;

public class TrivialLUTRobot extends QLearningRobot {
    public TrivialLUTRobot() {
        super(createLearning());
    }


    public static ILearning createLearning() {
        IActionRepresentation actionRepresentation = new MoveRepresentation
            ();
        IStateRepresentation stateRepresentation = new
            CoordinatesRepresentation();
        IFunctionApproximation functionApproximation = new LUT("
            TrivialLUTRobot.obj", false);
        IPolicy policy = new GoTopRight();
        return new QLearning(
                stateRepresentation,
                actionRepresentation,
                new ConcatenationRepresentation(),
                policy,
                functionApproximation,
                0.8, 0.1, 0.8);
    }
}
```

Listing 70: robot/TrivialLUTRobot.java

```java
package autograd;


import org.junit.Assert;
import org.junit.Test;

public class VariableTest {

    @Test
    public void testAddition() {
        Assert.assertEquals(new Addition().apply(new Parameter(12), new
            Parameter(2.)).evaluate(), 14., 0.);
    }

    @Test
    public void testVariableEvaluation() {
        Assert.assertEquals(new Parameter(250).evaluate(), 250., 0);
    }

}
```

Listing 71: autograd/VariableTest.java

```java
package fa;
```

```
2
3  import org.junit.Assert;
4  import org.junit.Ignore;
5  import org.junit.Test;
6  import representation.IRepresentable;
7
8  public class TestFunctionApproximation {
9      @Ignore
10     @Test
11     public void TestLUT() {
12         LUT lut = new LUT(null, true);
13         IRepresentable desiredState = new IRepresentable() {
14             @Override
15             public double[] toVector() {
16                 return new double[0];
17             }
18         };
19         IRepresentable otherState = new IRepresentable() {
20             @Override
21             public double[] toVector() {
22                 return new double[0];
23             }
24         };
25         double[] desiredResponse = new double[] {10};
26         double[] otherDesiredResponse = new double[] {5};
27         lut.train(desiredState, desiredResponse);
28         Assert.assertArrayEquals(desiredResponse, lut.eval(desiredState),
               0.);
29         lut.train(otherState, otherDesiredResponse);
30         Assert.assertArrayEquals(otherDesiredResponse, lut.eval(otherState)
               , 0.);
31         lut.train(desiredState, otherDesiredResponse);
32         Assert.assertArrayEquals(otherDesiredResponse, lut.eval(
               desiredState), 0.);
33     }
34 }
```

Listing 72: fa/TestFunctionApproximation.java

```
1  package nn;
2
3  import autograd.Parameter;
4  import jdk.jshell.spi.ExecutionControl;
5  import org.junit.Assert;
6  import org.junit.Test;
7
8  import java.util.Arrays;
9
10 public class NeuralNetworkTest {
11
12     @Test
13     public void testNeuralNetworkFactory() {
14         var model = Factory.createNeuralNetwork(new int[]{2, 4, 1}, new
               Sigmoid());
15         var result = model.evaluate(new double[]{0, 0});
16         Assert.assertEquals(result.length, 1);
17     }
18
```

```
19      @Test
20      public void testNeuralNetworkGradient() throws ExecutionControl.
            NotImplementedException {
21          var model = Factory.createNeuralNetwork(new int[]{2, 4, 1}, new
                Sigmoid());
22          Parameter[] parameters = model.getTrainableParameters();
23          for (Parameter parameter :
24                  parameters) {
25              parameter.setValue(1);
26          }
27          var result = model.evaluate(new double[]{1, 0});
28          double[] desired = new double[]{1};
29          Assert.assertEquals(result.length, 1);
30          double delta = 1e-5;
31          double expected = 0.9892621636390686; // obtained by pytorch
32          Assert.assertEquals(result[0], expected, delta);
33          var loss = new MinimumSquaredError(model.getOutput());
34          loss.setDesired(desired);
35          loss.backward(parameters, 1);
36          var gradients = new double[parameters.length];
37          for (int i = 0; i < parameters.length; i++) {
38              gradients[i] = parameters[i].getGradient();
39          }
40
41          Assert.assertArrayEquals(Arrays.stream(new double[]{ // calculated
                by pytorch
42                  -0.000114063048386015, -0.00010046639363281429,
                        -0.00010046639363281429, -0.00010046639363281429,
                        -0.00010046639363281429, -1.197589335788507e-05,
                        -1.197589335788507e-05, -1.197589335788507e-05,
                        -1.197589335788507e-05, -1.197589335788507e-05,
                        -1.197589335788507e-05, -0.0, -0.0, -0.0, -0.0
43          }).sorted().toArray(), Arrays.stream(gradients).sorted().toArray(),
                delta);
44      }
45
46      @Test
47      public void testNeuralNetworkGradientBipolar() throws ExecutionControl.
            NotImplementedException {
48          var model = Factory.createNeuralNetwork(new int[]{2, 4, 1}, new
                BipolarSigmoid());
49          Parameter[] parameters = model.getTrainableParameters();
50          for (Parameter parameter :
51                  parameters) {
52              parameter.setValue(1);
53          }
54          var result = model.evaluate(new double[]{1, -1});
55          double[] desired = new double[]{1};
56          Assert.assertEquals(result.length, 1);
57          double delta = 1e-5;
58          double expected = 0.8904789686203003; // obtained by pytorch
59          Assert.assertEquals(expected, result[0], delta);
60          var loss = new MinimumSquaredError(model.getOutput());
61          loss.setDesired(desired);
62          loss.backward(parameters, 1);
63          var gradients = new double[parameters.length];
64          for (int i = 0; i < parameters.length; i++) {
```

```
            gradients[i] = parameters[i].getGradient();
        }

        Assert.assertArrayEquals(Arrays.stream(new double[]{ // calculated by pytorch
                -0.011338012292981148, -0.005239490419626236,
                    -0.005239490419626236, -0.005239490419626236,
                    -0.005239490419626236, -0.004458376672118902,
                    -0.004458376672118902, -0.004458376672118902,
                    -0.004458376672118902, -0.004458376672118902,
                    -0.004458376672118902, -0.004458376672118902,
                    -0.004458376672118902, 0.004458376672118902,
                    0.004458376672118902, 0.004458376672118902,
                    0.004458376672118902
        }).sorted().toArray(), Arrays.stream(gradients).sorted().toArray(),
            delta);
    }
}
```

Listing 73: nn/NeuralNetworkTest.java

```
package optimization;

import autograd.UniformInitializer;
import dataset.BinaryToBipolarWrapper;
import dataset.XORBinaryDataSet;
import jdk.jshell.spi.ExecutionControl;
import nn.*;
import org.junit.Assert;
import org.junit.Ignore;
import org.junit.Test;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Optional;

public class GradientDescentTest {

    private final static int trials = 300;

    @Ignore
    @Test
    public void TestSimpleGD() throws ExecutionControl.
        NotImplementedException {
        var model = Factory.createNeuralNetwork(
                new int[]{2, 4, 1},
                new Sigmoid(),
                new UniformInitializer(-0.5, 0.5)
        );
        var dataSet = new XORBinaryDataSet();
        var optimizer = new GradientDescent(0.2, 0.);
        var loss = new MinimumSquaredError(model.getOutput());
        double finalLoss = model.fit(dataSet, optimizer, loss, 40000, 0.05)
            ;
        Assert.assertTrue("Big loss " + finalLoss, finalLoss < 0.05);
    }
```

```
36
37      @Ignore("Skipping slow convergence tests.")
38      @Test
39      public void TestConvergence() throws ExecutionControl.
            NotImplementedException, IOException {
40          int diverged = 0;
41          ArrayList<ConvergenceCollector> stats = new ArrayList<>();
42          for (int i = 0; i < GradientDescentTest.trials; i++) {
43              var model = Factory.createNeuralNetwork(
44                      new int[]{2, 4, 1},
45                      new Sigmoid(),
46                      new UniformInitializer(-0.5, 0.5)
47              );
48              var dataSet = new XORBinaryDataSet();
49              var optimizer = new GradientDescent(0.2, 0.);
50              var loss = new MinimumSquaredError(model.getOutput());
51              var collector = new ConvergenceCollector();
52              double finalLoss = model.fit(dataSet, optimizer, loss, 40000,
                    0.05, collector);
53              stats.add(collector);
54              if (finalLoss > 0.05) {
55                  diverged += 1;
56              }
57          }
58          outputGraphData("a", stats);
59          Assert.assertTrue("Convergence with high probability busted!",
                diverged < 6);
60      }
61
62      private void outputGraphData(String assignmentPart, ArrayList<
            ConvergenceCollector> stats) throws IOException {
63          FileWriter of = new FileWriter("doc/" + assignmentPart + "_avg.tex"
                );
64          double average = stats.stream().mapToInt(ConvergenceCollector::
                getEpochs).average().getAsDouble();
65          of.write(String.valueOf(average));
66          of.close();
67
68          Optional<ConvergenceCollector> representative = stats.stream().min(
                Comparator.comparingDouble(c -> Math.abs(c.getEpochs() - average
                )));
69          of = new FileWriter("doc/" + assignmentPart + ".tex");
70          of.write(representative.get().toString());
71          of.close();
72      }
73
74      @Ignore("Skipping slow convergence tests.")
75      @Test
76      public void TestBipolarGD() throws ExecutionControl.
            NotImplementedException, IOException {
77          int diverged = 0;
78          int trials = GradientDescentTest.trials;
79          ArrayList<ConvergenceCollector> stats = new ArrayList<>();
80          for (int i = 0; i < trials; i++) {
81              var model = Factory.createNeuralNetwork(
82                      new int[]{2, 4, 1},
83                      new BipolarSigmoid(),
84                      new UniformInitializer(-0.5, 0.5)
```

```
            );
            var dataSet = new BinaryToBipolarWrapper(new XORBinaryDataSet()
                );
            var optimizer = new GradientDescent(0.2, 0.);
            var loss = new MinimumSquaredError(model.getOutput());
            var collector = new ConvergenceCollector();
            double finalLoss = model.fit(dataSet, optimizer, loss, 3500,
                0.05, collector);
            if (finalLoss > 0.05) {
                diverged += 1;
            }
            stats.add(collector);
        }
        outputGraphData("b", stats);
        Assert.assertTrue("Convergence with high probability busted! " +
            diverged + " failure out of " + trials, diverged < 6);
    }

    @Ignore
    @Test
    public void TestBipolarMomentumGD() throws ExecutionControl.
        NotImplementedException, IOException {

        int diverged = 0;
        int trials = GradientDescentTest.trials;
        ArrayList<ConvergenceCollector> stats = new ArrayList<>();
        for (int i = 0; i < trials; i++) {
            var model = Factory.createNeuralNetwork(
                    new int[]{2, 4, 1},
                    new BipolarSigmoid(),
                    new UniformInitializer(-0.5, 0.5)
            );
            var dataSet = new BinaryToBipolarWrapper(new XORBinaryDataSet()
                );
            var optimizer = new GradientDescent(0.2, 0.9);
            var loss = new MinimumSquaredError(model.getOutput());
            var collector = new ConvergenceCollector();
            double finalLoss = model.fit(dataSet, optimizer, loss, 1000,
                0.05, collector);
            if (finalLoss > 0.05) {
                diverged += 1;
            }
            stats.add(collector);
        }
        outputGraphData("c", stats);
        Assert.assertTrue("Convergence with high probability busted! " +
            diverged + " failure out of " + trials, diverged < 6);
    }
}
```

Listing 74: optimization/GradientDescentTest.java

```
package rl;

import org.junit.Ignore;
import policy.IPolicy;
import representation.*;
import org.junit.Assert;
```

```java
import org.junit.Test;
import robocode.Robot;
import robocode.control.BattleSpecification;
import robocode.control.BattlefieldSpecification;
import robocode.control.RobocodeEngine;
import robocode.control.RobotSpecification;
import robocode.control.events.BattleAdaptor;
import robocode.control.events.RoundEndedEvent;
import robocode.control.events.TurnEndedEvent;
import robocode.control.snapshot.IRobotSnapshot;
import robot.TrivialLUTRobot;
import robot.TrivialLUTRobotConfident;

import java.io.File;
import java.util.ArrayList;

public class TestQLearning {

    @Ignore("Focus on testing TNinety")
    @Test
    public void TestTrivialLUTRobot() {
        new File("TrivialLUTRobot.obj").deleteOnExit();
        Robot opponent = new TrivialLUTRobot();
        ArrayList<IState> states = new ArrayList<>();
        TrivialLUTRobotConfident robot = new TrivialLUTRobotConfident();
        System.setProperty("NOSECURITY", "true");
        RobocodeEngine.setLogMessagesEnabled(false);
        RobocodeEngine engine = new RobocodeEngine(new java.io.File(System.
            getProperty("user.home") + "/robocode/"));
        engine.addBattleListener(new BattleAdaptor() {
            @Override
            public void onTurnEnded(TurnEndedEvent event) {
                super.onTurnEnded(event);
                for (IRobotSnapshot robotSnapshot: event.getTurnSnapshot().
                    getRobots()) {
//                    System.out.println(robotSnapshot.getShortName());
                    if (robotSnapshot.getShortName().equals("
                        TrivialLUTRobotConfident*")) {
                        states.add(new Coordinates(
                                (int) (robotSnapshot.getX() / 100),
                                (int) (robotSnapshot.getY() / 100), 0));
                    }
                }
            }
        });
//        engine.setVisible(true);
        int numberOfRounds = 100;
        BattlefieldSpecification battlefield = new BattlefieldSpecification
            (800, 600);
        RobotSpecification[] selectedRobots = engine.getLocalRepository(
            robot.getClass().getCanonicalName() + "*," + opponent.getClass()
            .getCanonicalName() + "*");
        BattleSpecification battleSpec = new BattleSpecification(
            numberOfRounds, battlefield, selectedRobots);
        engine.runBattle(battleSpec, true); // waits till the battle
            finishes
        engine.close();

```

```
57          Assert.assertTrue("Seemingly battle didn't happen. No state is
                collected.", states.size() > 1);
58          IState lastRun = states.get(states.size() - 1);
59          IState firstRun = states.get(0);
60          IPolicy policy = robot.getLearning().getPolicy();
61  //          double initialReward = policy.getReward(firstRun);
62  //          double finalReward = policy.getReward(lastRun);
63  //          Assert.assertTrue(
64  //                  String.format("Learning wasn't effective, initial reward
        %f, final reward %f", initialReward, finalReward),
65  //                  initialReward <= finalReward);
66      }
67  }
```

Listing 75: rl/TestQLearning.java

```
1   package robot;
2
3   import fa.LUT;
4   import org.junit.Test;
5   import representation.IState;
6   import robocode.Robot;
7   import robocode.control.BattleSpecification;
8   import robocode.control.BattlefieldSpecification;
9   import robocode.control.RobocodeEngine;
10  import robocode.control.RobotSpecification;
11  import robocode.control.events.BattleAdaptor;
12  import robocode.control.events.BattleCompletedEvent;
13
14  import java.io.File;
15  import java.io.FileWriter;
16  import java.io.IOException;
17  import java.util.ArrayList;
18  import java.util.Objects;
19
20  public class TestTNinetyRobot {
21
22      @Test
23      public void TestTrivialLUTRobot() {
24  //          testRobot(new LUTTNinetyRobot0(), 1, 100);
25  //          testRobot(new LUTTNinetyRobotOnline(), 1, 100);
26  //          testRobot(new LUTTNinetyRobotTerminal(), 500, 100);
27          testRobot(new LUTTNinetyRobot05(), 1, 100);
28  //          testRobot(new LUTTNinetyRobot(), 1, 100);
29      }
30
31      private void testRobot(Robot trainRobot, int step, int epochs) {
32
33          String outputFileName = "doc/" + trainRobot.getClass().getName() +
                ".tex";
34  //          new File(outputFileName).deleteOnExit();
35          new File("LUTTNinetyRobot.obj").deleteOnExit();
36          ArrayList<IState> states = new ArrayList<>();
37          LUTTNinetyRobotConfident testRobot = new LUTTNinetyRobotConfident()
                ;
38          Corners opponent = new Corners();
39          System.setProperty("NOSECURITY", "true");
40          RobocodeEngine.setLogMessagesEnabled(false);
```

```
41          RobocodeEngine engine = new RobocodeEngine(new File(System.
                getProperty("user.home") + "/robocode/"));
42          engine.addBattleListener(new BattleAdaptor() {
43              @Override
44              public void onBattleCompleted(BattleCompletedEvent event) {
45                  super.onBattleCompleted(event);
46                  boolean shouldPrint = false;
47                  try {
48                      FileWriter of = new FileWriter(outputFileName, true);
49                      for (var result :
50                              event.getIndexedResults()) {
51                          if (Objects.equals(result.getTeamLeaderName(),
                                testRobot.getClass().getCanonicalName() + "*"))
                                {
52                              of.write(result.getFirsts() + " ");
53                              System.out.println(result.getFirsts());
54                              shouldPrint = true;
55                          }
56                      }
57                      if (shouldPrint) {
58                          of.write(((LUT) (new LUTTNinetyRobot().getLearning
                                ().getFunctionApproximation())).getSize() + "\n"
                                );
59                      }
60                      of.close();
61                  } catch (IOException e) {
62                      e.printStackTrace();
63                  }
64              }
65          });
66          for (int i = 0; i < epochs; i++) {
67              int numberOfRounds = step;
68              BattlefieldSpecification battlefield = new
                    BattlefieldSpecification(800, 600);
69              Robot robot;
70              if (i % 2 == 0) {
71                  engine.setVisible(false);
72                  robot = trainRobot;
73              } else {
74                  numberOfRounds = 100;
75                  engine.setVisible(false);
76                  robot = testRobot;
77              }
78              RobotSpecification[] selectedRobots = engine.getLocalRepository
                    (robot.getClass().getCanonicalName() + "*," + opponent.
                    getClass().getCanonicalName() + "*");
79              BattleSpecification battleSpec = new BattleSpecification(
                    numberOfRounds, battlefield, selectedRobots);
80              engine.runBattle(battleSpec, true); // waits till the battle
                    finishes
81          }
82          engine.close();
83      }
84  }
```

Listing 76: robot/TestTNinetyRobot.java