MEDWIN PUBLISHERS
Committed to Create Value for Researchers

Petroleum & Petrochemical Engineering Journal
ISSN: 2578-4846

APPENDIX

Java Program for penetration rate prediction with neural networks:

```
package org.deeplearning4j.examples.feedforward.regression;
import org.datavec.api.records.reader.RecordReader;
import org.datavec.api.records.reader.impl.csv.CSVRecordReader;
import org.datavec.api.split.FileSplit;
import org.apache.commons.math3.analysis.function.Exp;
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.deeplearning4j.eval.RegressionEvaluation;
import org.deeplearning4j.nn.api.Model;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.deeplearning4j.datasets.iterator.impl.ListDataSetIterator;
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.Updater;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.deeplearning4j.optimize.listeners.ScoreIterationListener;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.dataset.api.preprocessor.NormalizerMinMaxScaler;
import org.nd4j.linalg.factory.Nd4j;
import org.nd4j.linalg.lossfunctions.LossFunctions;
import org.deeplearning4j.util.ModelSerializer;
import java.io.File;
import org.nd4j.linalg.dataset.api.preprocessor.serializer.NormalizerMinMaxScalerSerializer;
import java.util.Collections;
import java.util.List;
import java.util.Random;

/**
 * Created by Anwar on 3/15/2016.
 * An example of regression neural network for performing addition
 */
public class Regression {
   //Random number generator seed, for reproducability
   public static final int seed = 12345;
   //Number of iterations per minibatch
   public static final int iterations = 1;
   //Number of epochs (full passes of the data)
   public static final int nEpochs = 50000;
   //Number of data points
   //Batch size: i.e., each epoch has nSamples/batchSize parameter updates
   public static final int batchSize = 64;
   //Network learning rate
   public static final double learningRate = 0.0065;
```

```
// The range of the sample data, data in range (0-1 is sensitive for NN, you can try other ranges and see how it effects the results
// also try changing the range along with changing the activation function
public static final Random rng = new Random(seed);
public static void main(String[] args) throws Exception {
    RecordReader rr = new CSVRecordReader();
            rr.initialize(new FileSplit(new File("C:\\Users\\Profein\\Desktop\\dl4j-examples\\dl4j-examples-master\\dl4j-examples\\src\\main\\resources\\regression\\trainset.csv")));
    DataSetIterator trainIter = new RecordReaderDataSetIterator(rr, batchSize, 0, 0, true);
    RecordReader rrTest = new CSVRecordReader();
     rrTest.initialize(new FileSplit(new File("C:\\Users\\Profein\\Desktop\\dl4j-examples\\dl4j-examples-master\\dl4j-examples\\src\\main\\resources\\regression\\testset.csv")));
    DataSetIterator testIter = new RecordReaderDataSetIterator(rrTest, batchSize, 0, 0, true);
    NormalizerMinMaxScaler normalizer = new NormalizerMinMaxScaler(0, 1);
    normalizer.fitLabel(true);
    int numInput = 15;
    int numOutputs = 1;
    int nHidden = 20;
    MultiLayerConfiguration net = new NeuralNetConfiguration.Builder()
      .seed(seed)
      .iterations(iterations)
      .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
      .learningRate(learningRate)
      .weightInit(WeightInit.XAVIER)
      .updater(Updater.NESTEROVS).momentum(0.8)
     .list()
     .layer(0, new DenseLayer.Builder().nIn(numInput).nOut(nHidden)
     .activation(Activation.RELU)
      .build())
      .layer(1, new OutputLayer.Builder(LossFunctions.LossFunction.MSE)
        .activation(Activation.IDENTITY)
        .nIn(nHidden).nOut(numOutputs).build())
      .pretrain(false).backprop(true).build()          ;
    normalizer.fit(trainIter);
    trainIter.setPreProcessor(normalizer);
    testIter.setPreProcessor(normalizer);
    //Create the network
    MultiLayerNetwork model = new MultiLayerNetwork(net);
    model.init();
    model.setListeners(new ScoreIterationListener(1));
    RegressionEvaluation evaluation = new RegressionEvaluation(1);
    DataSet trainData = trainIter.next();
    DataSet testData = testIter.next();
    for (int i = 0; i < nEpochs; i++) {
     model.fit(trainData);
      INDArray features = testData.getFeatureMatrix();
      INDArray labels = testData.getLabels();
      INDArray predicted = model.output(features, false);
       evaluation.eval(labels, predicted);
        System.out.println(evaluation.stats());}
        INDArray features = testData.getFeatureMatrix();
        INDArray labels = testData.getLabels();
        INDArray prediction = model.output(features,false);
        normalizer.revert(testData);
        normalizer.revertLabels(prediction);
```

```
        System.out.println(prediction+" "+labels);
      File locationToSave = new File("MyReg2.zip");      //Where to save the network. Note: the file is in .zip format      - can be opened externally
    boolean saveUpdater = true;   //Updater: i.e., the state for Momentum, RMSProp, Adagrad etc. Save this if you   want to train your network more in the future
       ModelSerializer.writeModel(model, locationToSave, saveUpdater);
       File tmpFile = new File ( "normalizers1.zip");
       NormalizerMinMaxScalerSerializer serializer = new NormalizerMinMaxScalerSerializer();
       serializer.write(normalizer, tmpFile);
       System.out.print("Saved bih");  }}
```

For the reinforcement learning, Three separate classes were written: One for modelling the present state of the bit, another representing the model of the drilling environment and the third for the solution algorithm for the agent.

Bitstate.

java package edu.brown.cs.burlap.tutorials.domain.simple;

```
import burlap.mdp.core.state.MutableState;
import burlap.mdp.core.state.StateUtilities;
import burlap.mdp.core.state.UnknownKeyException;
import burlap.mdp.core.state.annotations.DeepCopyState;
import java.util.Arrays;
import java.util.List;
import static edu.brown.cs.burlap.tutorials.domain.simple.DrillModel.*;
@DeepCopyState
public class BitState implements MutableState{
 public int y1;
 public int y2;
   public int y;
   public double TIME;
   public int n;
      private final static List<Object> keys =     Arrays.<Object>asList(DEPTH_IN,DEPTH_OUT,BIT_FOOTAGE,CUMM_TIME,BIT_
NUMBER);
   public BitState() {    }
   public BitState(int y1, int y2,int y, double TIME,int n) {
      this.y1 = y1;
      this.y2 = y2;
      this.y = y;
      this.TIME = TIME;
      this.n = n;    }
   @Override
   public MutableState set(Object variableKey, Object value) {
      if(variableKey.equals(DEPTH_IN)){
      this.y1 = StateUtilities.stringOrNumber(value).intValue();       }
      else if(variableKey.equals(DEPTH_OUT)){
         this.y2 = StateUtilities.stringOrNumber(value).intValue();       }
      else if(variableKey.equals(BIT_FOOTAGE)){
         this.y = StateUtilities.stringOrNumber(value).intValue();       }
      else if(variableKey.equals(CUMM_TIME)) {
         this.TIME = StateUtilities.stringOrNumber(value).doubleValue();       }
      else if(variableKey.equals(BIT_NUMBER)) {
         this.n = StateUtilities.stringOrNumber(value).intValue();       }
      else{
         throw new UnknownKeyException(variableKey);       }
      return this;    }
   public List<Object> variableKeys() {
      return keys;    }
   @Override
   public Object get(Object variableKey) {
      if(variableKey.equals(DEPTH_IN)){
         return y1;       }
      else if(variableKey.equals(DEPTH_OUT)){
         return y2;       }
      else if(variableKey.equals(BIT_FOOTAGE)){
         return y;       }
      else if(variableKey.equals(CUMM_TIME)) {
         return TIME;       }
      else if(variableKey.equals(BIT_NUMBER)) {
         return n;     }
      else
      throw new UnknownKeyException(variableKey);    }
```

```
   @Override
   public BitState copy() {
      return new BitState(y1,y2,y,TIME,n);    }
   @Override
   public String toString() {
      return StateUtilities.stateToString(this);    }}
```

DrillModel.java

```
package edu.brown.cs.burlap.tutorials.domain.simple;
import burlap.mdp.auxiliary.DomainGenerator;
import burlap.mdp.core.StateTransitionProb;
import burlap.mdp.core.TerminalFunction;
import burlap.mdp.core.action.Action;
import burlap.mdp.core.action.UniversalActionType;
import burlap.mdp.core.state.State;
import burlap.mdp.singleagent.SADomain;
import burlap.mdp.singleagent.environment.SimulatedEnvironment;
import burlap.mdp.singleagent.model.FactoredModel;
import burlap.mdp.singleagent.model.RewardFunction;
import burlap.mdp.singleagent.model.statemodel.FullStateModel;
import burlap.shell.visual.VisualExplorer;
import burlap.visualizer.Visualizer;
import org.datavec.api.records.reader.RecordReader;

import org.datavec.api.records.reader.impl.csv.CSVRecordReader;
import org.datavec.api.split.FileSplit;
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.dataset.api.preprocessor.NormalizerMinMaxScaler;
import org.deeplearning4j.util.ModelSerializer;
import org.nd4j.linalg.dataset.api.preprocessor.serializer.NormalizerMinMaxScalerSerializer;
import org.nd4j.linalg.factory.Nd4j;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
public class DrillModel implements DomainGenerator {
public static final String DEPTH_IN = “y1”;
   public static final String DEPTH_OUT = “y2”;
   public static final String BIT_FOOTAGE = “y”;
   public static final String CUMM_TIME = “TIME”;
   public static final String BIT_NUMBER = “n”;
   public static final String ACTION_CHANGE = “changebit”;
   public static final String ACTION_CONTINUE = “continue”;
int targetDepth ;
protected double [][] map = new double [][] {
{6319,6512, 6607
 ,6793          ,7358          , 7449          , 7570          ,7865          ,8106          ,8296          ,8409
,8615          ,8883          ,8988          ,9025          ,9285          ,9522          ,9897          ,9905
,10252         ,10364         ,10372         ,10394         ,10599         ,10639         ,10649         ,10691
,10714         ,10720         ,10763         ,10780         ,10786         ,10793         ,10871         ,10876
```

,10887 ,11014 ,11017 ,11085 ,11121 ,11180 }, {5,10 ,10 ,0 ,15 ,20 ,20 ,20 ,20 ,20 ,25 ,25 ,10 ,10 ,25 ,25 ,25 , 0 ,5 , 5 , 5 ,10 ,10 ,10 ,10 ,20 , 20 ,0 ,0 ,0 ,0 ,0 , 0 , 0 ,0 ,0 ,0 ,0 ,0 , 0 ,0 }, {10,15 ,15 ,20 ,15 ,20 ,22 ,22 ,25 ,25 , 30 , 30 , 15 , 25 ,30 ,30 ,30 ,2 ,10 , 10 ,10 ,15 ,15 ,15 , 15 , 30 ,30 ,30 ,60 ,60 ,35 ,30 ,50 ,30 ,60 ,30 ,35 ,40 ,40 ,40 ,40 }, {9.4,9.4 , 9.4 , 9.4 , 9.4 , 9.4 , 9.4 , 9.4 , 9.5 , 9.6 , 9.6 , 9.7 , 9.7 , 9.7 , 9.8 , 9.8 , 9.8 , 9.8 , 9.8 , 9.9 , 9.9 , 9.9 , 9.9 , 9.9 , 9.9 , 9.9 , 10.2 , 10 ,10 , 10 , 10 , 10.1 , 10 , 10 , 10 , 10 , 10.1 , 10.1 , 10.1 , 10.1 , 10.1 }, {70,70 , 70 , 70 , 70 , 70 , 70 , 70 , 70 , 70 , 70 , 70 , 70 , 75 , 75 , 75 , 67 , 68 , 68 , 70 , 70 ,65 , 65 , 60 , 60 , 70 , 75 , 73 , 73 , 73 , 73 , 73 , 75 , 75 , 75 , 75 , 72 , 72 , 72 , 72 , 72 }, {0,4 , 4 , 4 , 4 , 4 , 4.5 , 4.5 , 4 , 4 , 4 , 4 , 0 , 0 , 7 , 7 , 8 , 9 , 9 , 9 , 9 ,9 , 9 , 9 , 0 , 7.2 , 7.2 , 9 , 9 , 9 , 9 , 10 , 4 , 4 , 11 , 11 , 11 , 10 , 10 , 10 , 10 }, {5,4, 4 , 4 , 4 , 4 , 4.5 , 4.5 , 4 , 6 , 6 , 6 , 0 , 0 , 8 , 8 , 10 , 9.5 , 9.5 , 9 , 9 , 9 ,9 , 9 , 9 , 0 , 7.2 , 7.2 , 9 , 9 , 9 , 9 , 10 , 4 , 4 , 11 , 11 , 11 , 10 , 12 , 12 , 10 }, {0,0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 0 , 0 , 0 , 0 , 0 ,0 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 0 , 0 , 0 }, {250,250 , 250 , 260 , 260 , 260 , 270 , 270 , 285 , 285 , 285 , 290 , 300 , 300 , 310 , 310 , 310 , 320 , 320 , 320 , 320 ,320 , 320 , 320 , 320 , 320 , 320 , 325 , 325 , 325 , 325 , 325 , 325 , 325 , 325 , 325 , 325 , 325 , 325 , 325 , 325 }, {250,245 , 245 , 255 , 255 , 255 , 265 , 265 , 275 , 275 , 275 , 280 , 280 , 280 , 300 , 300 , 280 , 290 , 290 , 290 , 290 ,290 , 290 , 290 , 290 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 , 280 }, {250,250 , 250, 260 , 260 , 260 , 260 , 270 , 270 , 280 , 280 , 280 , 285 , 285 , 285 , 295 , 295 , 290 , 300 , 300 , 300 , 300 ,300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 , 300 }, {40,40 , 40 , 40 , 40 , 40 , 40 , 40 , 40 , 40 , 40 , 40 , 40 , 0 , 0 , 40 , 40 , 40 , 40 , 40 , 40 , 40 ,40 , 40 , 40 , 0 , 40 , 40 , 60 , 50 , 40 , 40 , 60 , 40 , 0 , 40 , 40 , 40 , 40 , 52.5 , 52.5 , 40 }, {0,0 , 95 , 281 , 846 , 842 , 0 , 295 , 536 , 0 , 113 , 319 , 527 , 0 , 37 , 297 , 534 , 0 , 8 , 355 , 467 , 475 ,497 , 702 , 742 , 0 , 42 , 65 , 71 , 114 , 131 , 137 , 0 , 78 , 83 , 94 , 221 , 224 , 292 , 328 , 387 } };

```
public void setGoalLocation(int targetDepth){
   this.targetDepth = targetDepth;    }
@Override
public SADomain generateDomain() {
SADomain domain = new SADomain();
domain.addActionTypes(
 new UniversalActionType(ACTION_CHANGE),
  new UniversalActionType(ACTION_CONTINUE));
   DrillStateModel smodel = new DrillStateModel();
   RewardFunction rf = new ExampleRF(this.targetDepth);
   TerminalFunction tf = new ExampleTF(this.targetDepth);
   domain.setModel(new FactoredModel(smodel, rf, tf));
   return domain;    }
protected class DrillStateModel implements FullStateModel {
   @Override
   public List<StateTransitionProb> stateTransitions(State s, Action a) {
      BitState gs = (BitState) s;
      BitState bs = (BitState) s;
      int Depth_in = gs.y1;
      double time = gs.TIME;
      int footage = gs.y;
      int curDepth = gs.y2;
      int footagee = bs.y;
      int Depth_inn = bs.y1;
      int act = actionDir(a);
      int bitnum = bs.n;
      int[][] T = new int[2][2];
      T[0][0]=T[1][1]=1;
      T[0][1]=T[1][0]=0;
     List<StateTransitionProb> tps = new ArrayList<>(2);
      StateTransitionProb noChange = null;
      for (int i = 0; i < 2; i++) {
         double[] newPos = this.moveResult(Depth_in, curDepth, footage, Depth_inn, footagee, i,bitnum);
         if (newPos[1] != curDepth) {
            //new possible outcome
            BitState ns = gs.copy();
            ns.y1 = (int)newPos[0];
            ns.y2 = (int)newPos[1];

            ns.y = (int)newPos[2];
            ns.TIME = newPos[3]+time;
            ns.n = (int)newPos[4];
            tps.add(new StateTransitionProb(ns, T[i][act]));             }
         else{
            if(noChange != null){
               noChange.p += T[i][act];                }
          else{
            //otherwise create this new state and transition
          noChange = new StateTransitionProb(s.copy(), T[i][act]);
          tps.add(noChange);              }            }         }
      return tps;       }
   @Override
   public State sample(State s, Action a) {
      s = s.copy();
```

```
    BitState gs = (BitState) s;
    BitState bs = (BitState) s;
    int Depth_in = gs.y1;
    double time = gs.TIME;
    int footage = gs.y;
    int curDepth = gs.y2;
    int action = actionDir(a);
    int footagee = bs.y;
    int bitnum = bs.n;
    int Depth_inn = bs.y1;
    double[] newPos = this.moveResult(Depth_in, curDepth, footage, Depth_inn, footagee, action,bitnum);
    //set the new position
    gs.y1 = (int)newPos[0];
    gs.y2 = (int)newPos[1];
    gs.y = (int)newPos[2];
    gs.TIME = newPos[3]+time;
    gs.n = (int)newPos[4];
    //return the state we just modified
    return gs;      }
  protected int actionDir(Action a) {
    int adir = -1;
    if (a.actionName().equals(ACTION_CONTINUE)) {
      adir = 0;
    } else if (a.actionName().equals(ACTION_CHANGE)) {
      adir = 1;          }
    return adir;       }
  protected double[] moveResult(int y1, int y2, int y, int y3, int y4, int action,int n) {
    double ROP;
    double[] p = new double[13];
    double[]param ;
    for(int i = 0;i<map[0].length;i++) {
      if (y3 == map[0][i]) {
        for (int j = 0; j < 13; j++) {
          p[j] = map[j][i];}
          y2=(int)map[0][i+1];
        break;             }          }
    param = new double[]{p[1], p[2], p[11], p[3], p[4], y1, y2, 12.25, y, p[5], p[6], p[7], p[8], p[9],p[10]};
        double PR = estimate(param);
        if(PR<=0){
          PR=0.7;                }
    if (action == 0) {
      ROP = ((y2 - y3) / PR);
      for (int i = 0; i < map[0].length; i++) {
        if (y2 == map[0][i]) {
          y = y + y2 - y1;
          y1 = y2;
          break;                }             }
       } else {
      ROP = .2 + ((y2 - y3) / PR);


      n++;
      y = 0;
      y1 = y2;          }
```

```
        return new double[]{y1, y2, y, ROP,n};       }
    protected int estimate(double[]param) {
        int ROP = 0;
        try {
            File location = new File("MyReg2.zip");
            File norms = new File("normalizers.zip");
            NormalizerMinMaxScalerSerializer serializer = new NormalizerMinMaxScalerSerializer();
            NormalizerMinMaxScaler normalizer = serializer.restore(norms);
            MultiLayerNetwork net = ModelSerializer.restoreMultiLayerNetwork(location);
            final INDArray input = Nd4j.create(param);
            final INDArray output = Nd4j.create(new double[]{0});
            normalizer.transform(input);
            INDArray result = net.output(input);
            normalizer.revertLabels(result);
            ROP = result.getInt(0);
            return ROP;
        } catch (IOException e) {
            e.printStackTrace();         }
        return ROP;      }   }
  public static class ExampleRF implements RewardFunction {
    int targetDepth;
    public ExampleRF(int targetDepth){
      this.targetDepth=targetDepth;       }
    @Override
    public double reward(State s, Action a, State sprime) {
      BitState gs = (BitState) s;
      if(gs.y2==11121 ) {
        return -(gs.n *50000 + 7000*gs.TIME);          }
      else return 0;       }   }
  public static class ExampleTF implements TerminalFunction {
    int targetDepth;
    public ExampleTF(int targetDepth){
      this.targetDepth = targetDepth;       }
    @Override
    public boolean isTerminal(State s) {
      //get location of agent in next state
      int y2 = (Integer)s.get(DEPTH_OUT);
      //are they at goal location?
      if(y2 == this.targetDepth ){
        return true;          }
      return false;       }   } }
```

Program bitsolve

```
package edu.brown.cs.burlap.tutorials.domain.simple;
//import burlap.behavior.policy.Policy;
//import burlap.behavior.policy.PolicyUtils;
import burlap.behavior.singleagent.Episode;
//import burlap.behavior.singleagent.auxiliary.performance.LearningAlgorithmExperimenter;
//import burlap.behavior.singleagent.auxiliary.performance.PerformanceMetric;
//import burlap.behavior.singleagent.auxiliary.performance.TrialMode;
import burlap.behavior.singleagent.learning.LearningAgent;
//import burlap.behavior.singleagent.learning.LearningAgentFactory;
import burlap.behavior.singleagent.learning.tdmethods.QLearning;
//import burlap.behavior.singleagent.learning.tdmethods.SarsaLam;
//import burlap.behavior.singleagent.planning.Planner;
```

```
//import burlap.behavior.singleagent.planning.deterministic.DeterministicPlanner;
//import burlap.behavior.singleagent.planning.deterministic.uninformed.bfs.BFS;
//import burlap.behavior.singleagent.planning.stochastic.valueiteration.ValueIteration;
//import burlap.mdp.auxiliary.stateconditiontest.StateConditionTest;
//import burlap.mdp.auxiliary.stateconditiontest.TFGoalCondition;
//import burlap.mdp.core.TerminalFunction;

//import burlap.mdp.core.action.Action;
import burlap.mdp.core.state.State;
import burlap.mdp.singleagent.SADomain;
//import burlap.mdp.singleagent.common.GoalBasedRF;
import burlap.mdp.singleagent.environment.SimulatedEnvironment;
//import burlap.mdp.singleagent.model.FactoredModel;
//import burlap.mdp.singleagent.model.RewardFunction;
//import burlap.mdp.singleagent.oo.OOSADomain;
import burlap.statehashing.simple.SimpleHashableStateFactory;
//import org.datavec.api.records.writer.impl.csv.CSVRecordWriter;
import java.io.PrintWriter;
//import java.util.List;
public class bitsolve {
   public static void main(String [] args) throws Exception {
      DrillModel gw = new DrillModel();
      //PrintWriter writer  = new PrintWriter("exp2.csv");
      PrintWriter writer1  = new PrintWriter("exp6.csv");
      PrintWriter writer2  = new PrintWriter("exp7.csv");
      PrintWriter writer3  = new PrintWriter("exp8.csv");
      PrintWriter writer4  = new PrintWriter("exp9.csv");
      //TerminalFunction tf = new DrillModel.ExampleTF(8000);
      //RewardFunction rf = new DrillModel.ExampleRF(8000);
      //StateConditionTest goalCondition = new TFGoalCondition(tf);
      int[] depth = new int[]{6512,  6607          ,6793          ,7358          , 7449          , 7570          ,7865          ,8106
,8296          ,8409          ,8615          ,8883          ,8988          ,9025          ,9285          ,9522          ,9897
,9905          ,10252         ,10364         ,10372         ,10394         ,10599         ,10639         ,10649         ,10691
,10714         ,10720         ,10763         ,10780         ,10786         ,10793         ,10871         ,10876         ,10887
,11014         ,11017         ,11085         ,11121         ,11180};      gw.setGoalLocation(11180);
      //for(int i=0;i<80;i++){
        //depth[i]=100*(i+1);      //}
      final SADomain domain = gw.generateDomain();
      //ends when the agent reaches a location
      State initialState = new BitState(6319, 6512, 0,0.0,1);
      SimulatedEnvironment env = new SimulatedEnvironment(domain, initialState);
      //set up the state hashing system for looking up states
      final SimpleHashableStateFactory hashingFactory = new SimpleHashableStateFactory();
      //DeterministicPlanner planner = new BFS(domain, goalCondition, hashingFactory);
      //Policy p = planner.planFromState(initialState);
      //Planner planner = new ValueIteration(domain, 0.99, hashingFactory, 0.001, 1);
      //Policy p = planner.planFromState(initialState);
      //List<Action> actionSequence = PolicyUtils.rollout(p, initialState, domain.getModel()).actionSequence;
      //List<State> states = PolicyUtils.rollout(p,initialState,domain.getModel()).stateSequence;

      //System.out.println(actionSequence);
      //System.out.println(states);
      //System.out.println( PolicyUtils.rollout(p,initialState,domain.getModel()).numActions());
      //System.out.println( PolicyUtils.rollout(p,initialState,domain.getModel()).rewardSequence);
```

```
LearningAgent agent = new QLearning(domain, 0.99, hashingFactory, 10, 1);
//StringBuilder sb = new StringBuilder();
StringBuilder sb1 = new StringBuilder();
StringBuilder sb2 = new StringBuilder();
StringBuilder sb3 = new StringBuilder();
StringBuilder sb4 = new StringBuilder();
sb1.append("Counter");
sb1.append(',');
sb1.append("Cost");
sb1.append(',');
for(int i=0;i<depth.length-1;i++){
   sb1.append(depth[i]).append(" ft");
   sb1.append(',');       }
sb1.append('\n');
sb2.append("Counter");
sb2.append(',');


sb2.append("Cost");
sb2.append(',');
for(int i=0;i<depth.length-1;i++){
   sb2.append(depth[i]).append(" ft");
   sb2.append(',');       }
sb2.append('\n');
sb3.append("Counter");
sb3.append(',');
sb3.append("Cost");
sb3.append(',');
for(int i=0;i<depth.length-1;i++){
   sb3.append(depth[i]).append(" ft");
   sb3.append(',');       }
sb3.append('\n');
sb4.append("Counter");
sb4.append(',');
sb4.append("Cost");
sb4.append(',');
for(int i=0;i<depth.length-1;i++){
   sb4.append(depth[i]).append(" ft");
   sb4.append(',');       }
sb4.append('\n');
//run learning for 50 episodes
for (int i = 0; i < 200000; i++) {
   System.out.println(i);
   Episode e = agent.runLearningEpisode(env);
   Object[] actseq = e.actionSequence.toArray();
   //System.out.println(e.stateSequence);
   if(i>=150000 && i<160000){
      sb1.append(i);
      sb1.append(',');
      sb1.append("$").append(-1 * e.reward(40));
      sb1.append(',');
      for (int j=0;j<actseq.length;j++) {
         sb1.append(actseq[j]);
         sb1.append(',');            }
```

```
        sb1.append('\n');}
      if(i>=160000 && i<170000){
        if(i==150000){
        writer1.write(sb1.toString());
        writer1.close();}
        sb2.append(i);
        sb2.append(',');
        sb2.append("$").append(-1 * e.reward(40));
        sb2.append(',');
        for (Object anActseq : actseq) {
          sb2.append(anActseq.toString());
          sb2.append(',');            }
        sb2.append('\n');}
      if(i>=170000 && i<180000){
        if(i==170000){
        writer2.write(sb2.toString());
        writer2.close();}
        sb3.append(i);
        sb3.append(',');
        sb3.append("$").append(-1 * e.reward(40));
        sb3.append(',');
        for (Object anActseq : actseq) {
          sb3.append(anActseq.toString());
          sb3.append(',');             }
        sb3.append('\n');}
      if(i>=180000){
        if(i==180000){

        writer3.write(sb3.toString());
        writer3.close();}
        sb4.append(i);
        sb4.append(',');
        sb4.append("$").append(-1 * e.reward(40));
        sb4.append(',');
        for (Object anActseq : actseq) {
          sb4.append(anActseq.toString());
          sb4.append(',');            }
        sb4.append('\n');}
      //reset environment for next learning episode
      env.resetEnvironment();      }
    writer4.write(sb4.toString());
    writer4.close();
    System.out.println("Done lil bih");
    //LearningAgentFactory qLearningFactory = new LearningAgentFactory() {
      //public String getAgentName() {
        //return "Q-Learning";
      //}
      //public LearningAgent generateAgent() {
        //return new QLearning(domain, 0.99, hashingFactory, 10, 1);
      //}
    //};
    //LearningAlgorithmExperimenter exp = new LearningAlgorithmExperimenter(env, 1, 00000, qLearningFactory);
    //exp.setUpPlottingConfiguration(500, 250, 2, 1000,
        //TrialMode.MOST_RECENT_AND_AVERAGE,
```

```
        //PerformanceMetric.CUMULATIVE_STEPS_PER_EPISODE,
        //PerformanceMetric.AVERAGE_EPISODE_REWARD);
    //exp.startExperiment();
    //exp.writeStepAndEpisodeDataToCSV("expData");    }    }
```