```
package massim;

import java.util.HashMap;
import java.util.Map;

import javax.naming.CommunicationException;

/**
 * Agent.java
 * An abstract class for all the agents to be used in the
 * simulator
 *
 * @author Omid Alemi
 * @version 1.1 2011/10/06
 */
public abstract class Agent {

    /**
     * Agent Status Return Code AGCODE:
     *
     * OK                   Normal behavior of the agent while it is moving
     * ERR                  The sign of internal error within the agent
     * DONE                 Means the agent has reached the goal, but still is a
ctive and able to help
     * OFF                  Means the agent is not functioning anymore (no move,
 no communication, no help)
     */
    public static enum AGCODE { OK, ERR, DONE, OFF };

    private int id;
    private EnvAgentInterface env;

    private int[] actionCosts;

    // *** The beliefs
    // Personal beliefs (mental notes)
    private int resourcePoints = 0;
    private int rewardPoints = 0;
    // Percepts
    private RowCol pos;
    private RowCol myGoalPos;
    private RowCol[] agPos;
    // ****


    /**
     * The constructor
     *
     */
    public Agent(int id,EnvAgentInterface env) {
        this.id = id;
        this.env = env;
    }

    /**
     * Resets the agent’s internals to prepare it for a new run
     *
     * @param actionCosts               The action cost vector for this agent
     */
```

```
    public void reset(int[] actionCosts) {

        rewardPoints = 0;
        resourcePoints = 0;
        //path = new Path();

        this.actionCosts = new int[Environment.numOfColors];
        for (int i=0;i<Environment.numOfColors;i++)
            this.actionCosts[i] = actionCosts[i];

        // reset the agents positions beliefs
        agPos = new RowCol[Team.teamSize];
        for (int i=0;i<Team.teamSize;i++)
            this.agPos[i] = new RowCol(-1,-1);

        // reset the own positions
        pos = new RowCol(-1, -1);

        // reset the agent's  goal
        myGoalPos = null;

    }

    /**
     * Where agent performs its action.
     * No defualt action.
     * To be implemented by the customized agents.
     *
     * @return              AGCODE status code of current step
     */
    public AGCODE act() {

        return AGCODE.OK;
    }


    /**
     * Called by the Team in order to enable the agent to update its information

     * about the environment.
     *
     * We can pass all the information to the agent, but it can filer them
     * so that it can have partial observability.
     *
     * @param board                 The current state of the board
     * @param actionCostsMatrix     The action cost vectors of all the agents
     * @param goals                 The goals for all the agents
     * @param agentsPos             The current position of all the agents withi
n
     *                              the team
     */
    public void perceive(Board board, int[][] actionCostsMatrix, RowCol[] goals,
 RowCol[] agentsPos) {


        // Update the action cost vector
        for (int i=0;i<actionCostsMatrix[id].length;i++)
            this.actionCosts[i] = actionCostsMatrix[id][i];
```

```
        // Update the agents positions
        for (int i=0;i<agentsPos.length;i++)
            this.agPos[i] = agentsPos[i];

        // Update the own positions
        pos = agPos[id];

        // Update the agent's  goal
        myGoalPos = new RowCol(goals[id].row,goals[id].col);
    }

    /**
     * Sends all the outgoing messages, if any, in the current iteration
     * in the team step()
     */
    public void doSend() {
        // nothing as default
    }

    /**
     * Receives all the incoming message, if any, from other agents in
     * the current iteration in the team cycle
     */
    public void doReceive() {
        // nothing as default
    }


    /**
     *
     * @return              The id of the class
     */
    public int id() {
        return id;
    }

    /**
     * Returns the amount of reward points the agent owns at the moment
     *
     * @return              The amount of points the agent owns at the moment
     */
    public int rewardPoints() {
        return rewardPoints;
    }

    /**
     * The amount of resource points that the agent owns at the moment
     *
     * @return              The amount of resource points that the agent owns at
 the moment
     */
    public int resourcePoints() {
        return resourcePoints;
    }

    /**
     * Increases the reward points by the specified amount
     *
     * @param amount        The desired amount of points to be added
```

```
     */
    public void incRewardPoints(int amount) {
        rewardPoints += amount;
    }

    /**
     * Decreases the award points by the specified amount
     *
     * @param amount        The desired amount of points to be subtracted
     */
    public void decRewardPoints(int amount) {
        rewardPoints -= amount;
    }

    /**
     * Increases the resource points by the specified amount
     *
     * @param amount        The desired amount of points to be added
     */
    public void incResourcePoints(int amount) {
        resourcePoints += amount;
    }

    /**
     * Decreases the resource points by the specified amount
     *
     * @param amount        The desired amount of points to be subtracted
     */
    public void decResourcePoints(int amount) {
        resourcePoints -= amount;
    }

    /**
     * Enables the customized agents to get their position
     *
     * @return              The current position of the agent
     */
    public RowCol pos() {
        return pos;
    }

    /**
     * Enables the customized agents to access to the environment/agent interfac
e
     * of the team for communication and action
     *
     * @return              The instance of the
     */
    public EnvAgentInterface env() {
        return env;
    }

    /**
     * Enables the customized agents to get the position of their assigned goal
     *
     * @return              The positon of the goal
     */
    public RowCol goalPos() {
        return myGoalPos;
```

```
    }


    /**
     * Enables the customized agents to access their action costs vector
     *
     * @return              The action costs vector of the agent
     */
    public int[] actionCosts() {

        return actionCosts;
    }
}
```

```
package massim;

import java.util.HashMap;

import massim.Agent.AGCODE;

/**
 * Team.java
 *
 *
 * @author Omid Alemi
 * @version 1.1 2011/10/06
 */
public class Team {

    public static int teamSize;
    public static int calculationCost;
    public static int communicationCost;
    public static int achievementReward;
    public static int helpOverhead;

    private Agent[] agents;
    private Environment env;

    private int[][] actionCostMatrix;

    public static enum TeamStepCode {OK, DONE, ERR}

    /**
     * Default constructor
     */
    public Team() {
        env = new Environment();
        actionCostMatrix = new int[teamSize][Environment.numOfColors];
    }

    /**
     * Prepares the team for a new run by resetting its internal values
     *
     * @param agentsPos             The array of agents positions (initial posit
ions)
     * @param actionCostMatrix      The matrix of action costs for all the agent
s
     */
    public void reset(RowCol[] agentsPos, int[][]actionCostMatrix) {

        for (int i=0;i<teamSize;i++)
            env.setAgentPosition(i, agentsPos[i]);

        for (int i=0;i<teamSize;i++)
            for (int j=0;j<Environment.numOfColors;j++)
                this.actionCostMatrix[i][j] = actionCostMatrix[i][j];

        for (int i=0;i<teamSize;i++)
            agents[i].reset(actionCostMatrix[i]);
    }

    /**
     * Called by the simulation engine in each step of simulation
```

```
     *
     * @return ENDSIM                code if the simulation is over
     */
    public TeamStepCode step() {

        // 0. Update Agents Percepts

        for (int i=0;i<agents.length;i++)
            agents[i].perceive(Environment.board(), actionCostMatrix, Environmen
t.goals(), env.agentsPosition());


        // 1. Communication Phase

        int noMsgPass = 1;
        do {
            for (int i=0;i<teamSize;i++)
                agents[i].doSend();

            for (int i=0;i<teamSize;i++)
                agents[i].doReceive();

            if (env().communicationMedium().isEmpty())
                noMsgPass--;

        } while (!env().communicationMedium().isEmpty() || noMsgPass > 0);

        // 1. Action Phase

        boolean allDone = true; // this way of checking is just temporally and f
or tests
        for (int i=0;i<agents.length;i++)
            if (agents[i].act() != AGCODE.OFF)
                allDone = false;


        if (allDone)
            return TeamStepCode.DONE;
        else
            return TeamStepCode.OK;
    }

    /**
     * Enables the customized team classes to access the environment of the team
     *
     * @return              The instance of the team's environment
     */
    public Environment env() {
        return env;
    }

    /**
     * Enables the customize team classes to create and set their own agent type
s
     *
     * @param agents        The array of customized agent objects
     */
    public void setAgents(Agent[] agents) {
        this.agents = agents;
```

```
    }

    /**
     * Enables the customized team classes to get access to individual agents of
 the
     * team
     *
     * @param agent         The id of the desired agent
     * @return              The instance of the agnet object with the specified
id
     */
    public Agent agent(int agent) {
        return agents[agent];
    }

    /**
     * To get the collective resource points for the team
     *
     * @return              The amount of resources points that all the team's a
gents
     *                      own
     */
    public int teamResourcePoints() {
        int sum = 0;
        for (Agent a: agents)
            sum += a.resourcePoints();
        return sum;
    }

    /**
     * To get the collective reward points for the team
     *
     * @return              The amount of reward points that all the team's agen
ts own
     */
    public int teamRewardPoints() {
        int sum = 0;
        for (Agent a: agents)
            sum += a.rewardPoints();
        return sum;
    }
}
```

```
package massim;

/**
 * The Environment class
 *
 * @author Omid Alemi
 * @version 1.0 2011/10/06
 */
public class Environment implements EnvAgentInterface {

    public static int numOfColors; // This can be derived from the colorRange, b
ut for the sake of easier programming I'm keeping it like this!
    public static int[] colorRange;
//  public static int minActionCost;
//  public static int maxActionCost;
    public static int[] actionCostRange;


    public static double disturbanceLevel;
    public static double awarenessProb;

    private static Board mainBoard;
    private static RowCol[] goals;

    private RowCol[] agentsPosition;
    private CommMedium communicationMedium;

    /**
     * The constructor
     *
     */
    public Environment() {
        communicationMedium = new CommMedium();
        agentsPosition = new RowCol[Team.teamSize];
    }

    /**
     * Sets the board to a new representation
     *
     * @param newBoard              The desired new representation of the board
     */
    public static void setBoard(Board newBoard) {
        mainBoard = new Board(newBoard);
    }

    /**
     * Sets the positions of the goals
     *
     * @param newGoals              Array of goal positions
     */
    public static void setGoals(RowCol[] newGoals)  {
        goals = new RowCol[newGoals.length];

        for (int i=0;i<goals.length;i++)
            goals[i] = newGoals[i];
    }

    /**
     * Enables the access to the board's representation
```

```
     *
     * @return                      The instance of the board
     */
    public static Board board() {
        return mainBoard;
    }

    /**
     * Enables the access to the goals positions
     *
     * @return                      Array of goal positions
     */
    public static RowCol[] goals() {
        return goals;
    }


    /**
     * Returns the position of the specified agent
     *
     * @param agent                 The agent's id
     * @return                      The agent's position
     */
    public RowCol agentPosition(int agent) {
        return agentsPosition[agent];
    }

    /**
     * Returns the positions of all the agents with this environment instance
     *
     * @return                      Array of agents' positions
     */
    public RowCol[] agentsPosition() {
        return agentsPosition;
    }

    /**
     * Sets the position of an agent.
     *
     * @param agent                 The id of the agent
     * @param newPos                The desired position of the agent
     */
    public void setAgentPosition(int agent, RowCol newPos ) {

        agentsPosition[agent] = new RowCol(newPos.row,newPos.col);

    }

    /**
     * Enables the access to the communication medium for the agent
     *
     * @return                      The instance of the communication medium of
the environment
     */
    public CommMedium communicationMedium() {
        return communicationMedium;
    }

    /**
```

```
     * The move action of agents.
     * It can be called by an agent for its own move or for some other agent's m
ove (help).
     *
     *  @param agent                The id of the agent to be moved.
     *  @param newPos               The new position of the agent
     *  @return                     True if the action was successful / false ot
herwise
     */
    public boolean move(int agent, RowCol newPos) {

        if (!RowCol.areNeighbors(agentsPosition[agent], newPos))
                return false;

        agentsPosition[agent] = newPos;

        return true;
    }

    /**
     * Converts current state of this instance of environment to a string
     *
     * @return                  The string representation of the environment
     */
    @Override
    public String toString() {
        String s = "";
        s +=("\nThe Board\n");
        s +=(mainBoard.toString());
        s +=("\n");
        s +=("Agents Positions:\n");
        for (int i=0;i<agentsPosition.length;i++)
            s +="Agent "+i +":(" + agentsPosition[i].row +"," + agentsPosition[i].
col +")\n";
        s +=("\n");
        s +=("Communication Channels:\n");
        s +=(communicationMedium.toString());
        return s;
    }

    /**
     * Enables the agents to have the information about the range of action cost
s
     *
     * @return                  The action costs range in an array
     */
    @Override
    public int[] actionCostRange() {
        return actionCostRange;
    }


    /**
     * Enables the agents to have the information about the range of colors on t
he board
     *
     * @return                  The color range in an array
     */
    @Override
```

```
    public int[] colorRange() {
        return colorRange;
    }


}
```

```java
package massim;

/**
 * The interface of the environment that the agents can use to access the
 * environment
 *
 * @author Omid Alemi
 * @version 1.0 2011/10/06
 */
public interface EnvAgentInterface {
    public CommMedium communicationMedium();
    public boolean move(int agent, RowCol newPos);
    public int[] actionCostRange();
    public int[] colorRange();
}
```

```
package massim.agents;

import tests.DummyMessage;
import massim.Agent;
import massim.EnvAgentInterface;
import massim.Board;
import massim.Environment;
import massim.Goal;
import massim.Path;
import massim.RowCol;

public class DummyAgent extends Agent {

    private Board theBoard;
    private Path path;

    boolean sentHelpReq = false;
    boolean recHelpReq = false;
    boolean shouldAck = false;
    boolean reachedThere = false;

    public DummyAgent(int id, EnvAgentInterface env) {
        super(id,env);
        System.out.println("Hello from DummyAgent " + id());
    }


    @Override
    public void perceive(Board board, int[][] costVectors, RowCol[] goals, RowCo
l[] agentsPos) {

        super.perceive(board, costVectors, goals, agentsPos);

        theBoard = board;
        System.out.println("Agent " + id() +" New Percepts:");
        System.out.println("Agent " + id() +": resourcePoints = "+ resourcePoints());
        System.out.println("Agent " + id() +": my pos = "+ pos()
                +": my goal's pos = "+ goalPos());

        if (path == null && goals[id()] != null)
            findPath();

        if (pos().equals(goalPos()))
            reachedThere = true;
    }

    public int getCellCost(RowCol cell) {

        int [] colorRange = env().colorRange();
        int index = 0;
        for (int i=0;i<colorRange.length;i++)
        {
            int color = theBoard.getBoard()[cell.row][cell.col];
            if (color == colorRange[i])
                index = i;
        }

        return actionCosts()[index];
    }
```

```
    @Override
    public AGCODE act() {
        AGCODE code = AGCODE.OK;;

        if (!reachedThere)
        {
            RowCol nextPos = path.getNextPoint(pos());
            if (env().move(id(), nextPos))
                {
                    System.out.println("Agent " + id() +": moving to " + nextPos );

                    decResourcePoints(getCellCost(nextPos));
                }
            else
                System.out.println("Agent " + id() +": failed to move to " + nextPos );
        }
        else
        {
            code = AGCODE.OFF;
        }


        return code;
    }


    @Override
    public void doSend() {


    }

    @Override
    public void doReceive() {


    }

    private void findPath() {
        System.out.println("Agent " + id() +": Does not have a path, finding one ...");

        path = Path.getShortestPaths(pos(), goalPos(), theBoard.getBoard(), acti
onCosts(), 1).get(0);

        System.out.println("Agent " + id() +": My path will be: " + path);
    }

}
```

```
package massim.agents;

import massim.RowCol;
import massim.Team;

public class DummyTeam extends Team {


    public DummyTeam() {
        super();

        DummyAgent[] agents = new DummyAgent[teamSize];

        for (int i=0;i<teamSize;i++)
            agents[i] = new DummyAgent(i,env());

        setAgents(agents);

    }

    public void reset(RowCol[] agentsPos, int[][] actionCostsMatrix) {
        super.reset(agentsPos, actionCostsMatrix);

        for(int i=0;i<teamSize;i++)
            agent(i).incResourcePoints(1000);

    }
}
```

```
package tests;

import massim.Team.TeamStepCode;
import massim.agents.DummyAgent;
import massim.agents.DummyTeam;
import massim.*;

public class AgentTester {


    public static void main(String[] args) {

// Simulation-wide settings
        SimulationEngine.numOfTeams =1;
        Team.teamSize = 4;
        Environment.numOfColors = 5;
        Environment.colorRange = new int[] {10,11,12,13,14};
        Environment.actionCostRange = new int[] {10,15,20,30,50};


        DummyTeam dt = new DummyTeam();

// Experiment-wide settings
        Board board = Board.randomBoard(5, 5);

        RowCol[] goals = new RowCol[Team.teamSize];  // can be assigned randomiz
ed, etc;
        goals[0] = new RowCol(4,4);
        goals[1] = new RowCol(0,4);
        goals[2] = new RowCol(4,0);
        goals[3] = new RowCol(0,0);

        Environment.setBoard(board);
        Environment.setGoals(goals);

        int[][] actionCostsMatrix = {{20,10,10,15,50}, // can be assigned random
ized, etc;
                                     {10,10,50,20,10},
                                     {10,10,50,15,30},
                                     {15,30,20,10,10}};

        RowCol[] agentsPos = {new RowCol(0,0), // can be assigned randomized, et
c;
                              new RowCol(4,0),
                              new RowCol(0,4),
                              new RowCol(2,2)};

// Run-wide settings

        dt.reset(agentsPos,actionCostsMatrix);

// Run

        System.out.println("The initial env: "+dt.env());
        System.out.println("The initial team's resources = "+dt.teamResourcePoints());

        TeamStepCode tsc = TeamStepCode.OK;
        while (tsc == TeamStepCode.OK)
        {
```

```
            System.out.println("--------------------");
            tsc = dt.step();
        }

        System.out.println("The final team's resources = "+dt.teamResourcePoints());
    }

}
```

**Table of Contents**