```java
package massim;

/**
 * Agent.java
 * An abstract class for all the agents to be used in the
 * simulator
 *
 * @author Omid Alemi
 * @version 1.0 2011/10/01
 */
public abstract class Agent {

    private RowCol pos;
    private Team team;

    private RowCol[] myPath;  // Or possibly of type Path

    /**
     * Default constructor
     */
    public Agent() {

    }

    /**
     * Initializes the agent
     */
    public void init(Team t) {
        // set the attributes to their initial values
        setTeam(t);

    }


    /**
     * Where agent performs its action
     * @return 0 if it was successful, −1 for error (might not
     *           be the right place for this)
     */
    public int act() {
        // just move to the next position in the path as the
        // default action, maybe do nothing as default
        return 0;
    }


    /**
     * Called by the Team in order to enable the agent to update
     * its information about the environment
     * @param board The current state of the board
     * @param agentsPos The current position of the agent’s teammates
     *        on the board
     */
    public void perceive(Board board, RowCol[] agentsPos) {
        // Keep the necessary information private
    }

    /**
     * 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
     */
```

```java
    public void doReceive() {
        // nothing as default
    }

    /**
     * Sets the team that the agent belongs to
     * @param t The reference to the team
     */
    public void setTeam(Team t) {
        team = t;
    }
}
```

```java
package massim;

/**
 * The class to hold the board settings
 * @author Omid Alemi
 *
 */
public class Board {

    private int[][] mainBoard;

    private final int rows;
    private final int cols;


    /**
     * Constructor 1: just with the size
     *
     * @param r The number of rows of the board
     * @param c The number of columns of the board
     */
    public Board(int r, int c) {
        rows = r;
        cols = c;
    }

    /**
     * Constructor 2: get the board setting
     * @param board The 2dim array, representing the board’s initial
     *        setting
     */
    public Board(int[][] board) {
        rows = board.length;
        cols = board[0].length;
        // board −> mainBoard
    }

    /**
     * Sets the board setting to the inputBoard
     * @param initBoard The input board setting to be the main board setting
     */
    public void setBoard(int[][] inputBoard) {

    }

    /**
     * Returns the board setting
     * @return 2 dim array of int representing the board’s setting
     */
    public int[][] getBoard(){
        return mainBoard;
    }


    /**
     * Sets the value of one specific cell
     * @param row
     * @param col
     * @param color
     */
    public void  setCell(int row, int col, int color) {

    }

    /**
     * Returns a board with randomly filled values (colors).
     * @return A new instance of the Board class
     */
    public static Board randomBoard() {
```

```java
        return null;
    }

    /**
     * Adds random values (disturbance) to the cells of the board.
     * Each cell on the board may be changed based on the probability
     * defined by disturbanecLevel
     * @param disturbanceLevel The level of disturbance, between 0 and 1.0
     */
    public void distrub(double disturbanceLevel) {

    }

    /**
     * The overridden clone() method.
     * would be used to create a new copy of the current board’s representation.
     */
    @Override
    public Board clone() {
        // Creates a new instance of the Board class with the
        // same internal representation
        return null;
    }
}
```

```
package massim;

import java.util.HashMap;

/**
 * CommMedium.java
 * Responsible for all the communications within a team of
 * agents
 *
 * @author Omid Alemi
 * @version 1.0 2011/10/01
 */
public class CommMedium {

    private int numOfAgent;
    Agent[] agents;
    HashMap<Agent,String[]> buffers;

    /**
     * The default constructor
     */
    public CommMedium() {

    }

    /**
     * Puts the msg into the receiver’s special buffer for the sender
     * @param sender The sender agent
     * @param receiver The receiver agent
     * @param msg The message
     */
    public void send(Agent sender, Agent receiver, String msg) {

    }

    /**
     * Puts the msg into all the agent’s special buffer for the sender
     * @param sender The sender agent
     * @param msg The message
     */
    public void broadcast(Agent sender, String msg) {

    }

    /**
     * Returns the tuples of the <sender,msg> for all the incoming
     * messages for the receiver agent
     * @param receiver The receiver agent
     * @return Tuples of the <sender,msg>
     */
    public HashMap<Agent,String> receive(Agent receiver) {

        return null;
    }

    /**
     * To check whether the communication medium is empty. Means there
     * were no communication during the last iteration
     * @return true if all the buffers for all the agents are empty.
     *         false otherwise
     */
    public boolean isEmpty() {

        return false;
    }

}
```

```java
package massim;

/*
Colored Trails

Copyright (C) 2006−2007, President and Fellows of Harvard College.  All Rights R
eserved.

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110−1301, USA.
*/


/**
<b>Description</b>
All game board locations are described as row−column pairs, which are
represented by the RowCol class.  This class includes methods to determine
whether two locations are the same (’equals()’) and whether two locations
are adjacent to each other (’areNeighbors()’).

<p>

<b>Issues</b>
This class has two fields, representing a row and column on the board.
These are public fields and they lack ’get’ and ’set’ methods.
<p>
[depending upon how RowCol instances are used, we might want to think
about adding accessors and making the fields immutable (by making them ’final’)
once they are set in the constructor.]

<p>

<b>Future Development</b>
If we are interested to pursue other game board geometries, then more generic
ways of representing location need to be constructed.  Nevertheless, from the
point of view of an API, the ’equals’ and ’areNeighbors’ methods are already gen
eral.

<p>

<b>Original Summary</b>
* A type representing a (row,col) position, as opposed to (x,y).  All
* coordinates in CT3 should be in terms of (row,col).
*
* @author Paul Heymann (ct3@heymann.be)
@author Sevan G. Ficici (class−level review and comments)
*/


import java.io.Serializable;
import java.util.LinkedHashSet;


public class RowCol implements Serializable {
/** Rows */
public int row;
/** Columns */
public int col;
```

```java

public RowCol(int row, int col) {
    this.row = row;
    this.col = col;
}

public RowCol(RowCol rc) {
    this.row = rc.row;
    this.col = rc.col;
}

/**
    Move constructor (SGF)

    @param rc           starting point
    @param deltarow     change in row (may be negative)
    @param deltacol     change in column (may be negative)
*/
public RowCol(RowCol rc, int deltarow, int deltacol)
{
    this.row = rc.row + deltarow;
    this.col = rc.col + deltacol;
}

public String toString() {
    return "(" + row + "," + col + ")";
}

/**
 * Determine if this position equals another.
 * @param other The other position to compare to.
 * @return Whether the two RowCol objects refer to the same position.
 */
public boolean equals(Object o) {
    RowCol other = (RowCol) o;
    return (row == other.row && col == other.col);
}

/**
    Calculate distance from here to some other location (SGF)
*/
public int dist(RowCol other)
{
    return Math.abs(this.row − other.row) + Math.abs(this.col − other.col);
}

/**
 * Determine whether two (row,col) positions are neighbors.
 * @param rc1 The first position.
 * @param rc2 The second position.
 * @return Whether the positions are in fact neighbors.
 */
public static boolean areNeighbors(RowCol rc1, RowCol rc2) {
    return ((Math.abs(rc1.row − rc2.row) +
            Math.abs(rc1.col − rc2.col)) == 1);
}

/**
 * Gets the neighbor position of a position
 * @param board Board of the game
 * @return Neighbor positions
 * @author ilke
 */
public LinkedHashSet<RowCol> getNeighbors(Board board) {
    LinkedHashSet<RowCol> neighbors = new LinkedHashSet<RowCol>();
    /*
    if( col−1 >= 0 )
        neighbors.add(new RowCol(row, col−1));
    if( row−1 >= 0 )
```

```
        neighbors.add(new RowCol(row−1, col));
    if( col + 1 < board.getCols())
        neighbors.add(new RowCol(row, col+1));
    if( row + 1 < board.getRows())
        neighbors.add(new RowCol(row+1, col));
    */
    return neighbors;
}
}
```

```
/**
 *
 */
package massim;

/**
 * @author Omid Alemi
 * @version 1.0  2011/10/02
 */
public enum SimColor {
  RED, GREEN, BLUE, YELLOW, ORANGE, PINK, PURPLE, GREY;


}
```

```
package massim;

/**
 * SimParams.java
 * Purpose: The interface for the class holding different
 * costs and awards defined by the user of the simulator
 *          and used by the agents
 * @author Omid Alemi
 * @version 1.0 2011/10/01
 */

// x still not sure about using the interface ...
public interface SimParams {

}
```

```
package massim;

import java.util.List;

/**
 * The class representing the current state of the
 * simulator: the simulator counter, all the team states,
 * and the board status.
 *
 * @author Omid Alemi
 * @version 1.0 2011/10/02
 *
 */
public class SimState {

    TeamState[] teamsState;
    int simStep;
    Board board;

    /**
     * The constructor method.
     * @param simStep The simulator’s counter at the this specific moment
     * @param board The current board state
     */
    public SimState(int simStep, Board board) {
        //The board should be copied internally, not referenced.
    }

    /**
     * Adds a team state to the simulation’s state
     * @param ts The team state
     */
    public void addTeamState(TeamState ts) {

    }

    /**
     * Returns the simulator’s counter
     * @return Simulator’s counter
     */
    public int simStep() {
        return simStep;
    }

    /**
     * Returns the board representation of the simulator
     * stored at the time step simStep
     * @return The board object
     */
    public Board board() {
        return board;
    }

    /**
     *
     * @return The list of teamState object for all the teams
     * in the simulator.
     */
    public TeamState[] teamsState() {
        return teamsState;
    }

}
```

```
package massim;

import java.util.PriorityQueue;

/**
 * The Multiagent Teamwork Simulator
 * The main class of the simulator
 *
 * @author Omid Alemi
 * @version 1.0 2011/10/01
 *
 */
public class Simulator {

    private int simCounter; //change the name

    private Board board;
    private Team[] teams;

    public static enum SimStepCode {SIMOK, SIMEND, SIMERR}

    public static SimParams simParams;

    /**
     * Constructor
     *
     * @param teams The array of team to be participated in the simulation
     * @param simParams The class holding all the simulator parameters
     * @param initBoard The initial board settings
     */
    public Simulator(Team[] teams, SimParams simParams) {

    }

    /**
     * Initializes the simulator.
     * Should be called before step() method
     * @param initBoard The initial board, defined in the frontend
     */
    public void init(Board initBoard) {
        // load the initial board state into the board
        // initialize the teams
        // set the counter to zero
    }

    /**
     * Performs one step of the simulation based on the current simulator’s
     * counter.
     * Should be called by the frontend software
     * @param disturbanceLevel The level of desired disturbance on the board
     * @return The proper code from the SimStepCode enum
     */
    public SimStepCode step(double disturbanceLevel) {
        // increase the counter by 1
        // refresh the board: only add the disturbance
        // board.disturb(disturbanceLevel);
        // for each team in teams[]
        //      team.step(board);
        // return the proper simulation step code

        return SimStepCode.SIMOK;
    }

    /**
     * can be used by a frontend to run the simulator until the end
     * without interruption
     */
    public void autoplay() {
        // run the simulation from current step in a loop until the last step
```

```
        // (determined by the return code) without user interaction
    }


    /*
     * NOT SURE IF WE NEED THIS ANYMORE.
     * EVERYTHING CAN BE DONE USING THE SIMSTATE and TEAMSTATE CLASSES.
    public void finish() {
        // sum the scores
        // ********
    }*/


    /**
     * @return The current state of the simulation in a SimState object
     */
    public SimState getSimulationState() {

        return null;
    }
}
```

```java
package massim;

import java.util.HashMap;

/**
 * Team.java
 *
 *
 * @author Omid Alemi
 * @version 1.0 2011/10/01
 */
public class Team {

    private Agent[] agents;
    private CommMedium communicationMeduim;

    /**
     * Default constructor
     */
    public Team() {

    }

    /**
     * Constructor, getting a set of agents as the members of the team
     * @param agents An array of agents
     */
    public Team(Agent[] agents) {
        init(agents);
    }

    /**
     * Initializes the team object
     * @param agents Array of initial agents
     */
    public void init(Agent[] agents) {
        // communicationMeduim = new CommMedium();
        // this.agents = agents

        // for each agent a
        //     a.init(thisjjjj)

        // Set all the results to the initial values
    }

    /**
     * Called by the simulator in each step of simulation
     * @return ENDSIM code if the simulation is over
     */
    public int step(Board board) {

        // 0. Update Agents Percepts
        // agPos[] = position of all the agents in the team
        // for each agent a in agents[]
        //     a.perceive(board, agPos[])


        // 1. Communication Phase
        // noMsgPass = 5
        // do
        //     for each agent a in agents[]
        //         a.doSend();
        //     for each agent a in agents[]
        //         a.doReceive();

        // ?? WEHRE SHOULD WE GET THE STATUS CODE FROM THE AGENT?
        // (in doReceive or in a separate method just for that?)

        //     if (there was no communication)
```

```java
        //         noMsgPass--;
        //     else
        //         noMsgPass = 5;
        // while (there was at least one communication && noMsgPass != 0)


        // 1. Action Phase
        // for each agent a in agents[]
        //     a.act()

        return 0;
    }

    /**
     * Returns the team state, for the frontend usages.
     * team state should consist all the information about the agents’
     * position, resources, etc.
     * @return
     */
    public TeamState getTeamState() {

        return null;
    }

    /**
     * Calls the send method of the communication medium
     * @param sender The sender Agent
     * @param receiver The reciever Agent
     * @param msg The message
     */
    public void send(Agent sender, Agent receiver, String msg) {
        communicationMeduim.send(sender, receiver, msg);
        //or if the communicationMedium deals with the integers
        //instead of objects, the sender and receiver objects
        //can be mapped to their integer indices before calling
        //the method in communicationMedium
    }

    /**
     * Calls the receive method of the communication medium.
     * @param receiver The receiver agent
     * @return The list of all incoming messages for the receiver agent
     */
    public HashMap<Agent,String> receive(Agent receiver) {
        //or if the communicationMedium deals with the integers
        //instead of objects, the receiver objects
        //can be mapped to its integer indices before calling
        //the method in communicationMedium
        return communicationMeduim.receive(receiver);
    }

}
```

```java
package massim;

/**
 * Class to hold the information about the team.
 * All the agents' positions, resources, points,
 * etc.
 * This class is used by the frontends for representation
 * and post-processing purposes only.
 *
 * @author Omid Alemi
 * @version 1.0
 */
public class TeamState {

}
```

```
package massim.agents;

import massim.Agent;

public class MAPAgent extends Agent {

    // should have refrences to its teammates

    /**
     * The MAP agents perform their ’move’ action here
     */
    @Override
    public int act() {

        return 0;
    }

    @Override
    public void doSend() {
        // Do the MAP Protocol

        // team.send(this, agents[2], "1,2,bid:400")
        // or team.communicationMedium.send()
    }

    @Override
    public void doReceive() {
        // Do the MAP Protocol

        // HashMap<Agent,String> incoming = team.receive(this);
    }

}
```

Table of Contents

```
 1 Agent.java.......... sheets   1 to   1 ( 1) pages   1-  2   82 lines
 2 Board.java.......... sheets   2 to   2 ( 1) pages   3-  4   94 lines
 3 CommMedium.java..... sheets   3 to   3 ( 1) pages   5-  5   68 lines
 4 RowCol.java......... sheets   4 to   5 ( 2) pages   6-  8  146 lines
 5 SimColor.java....... sheets   6 to   6 ( 1) pages   9-  9   15 lines
 6 SimParams.java...... sheets   7 to   7 ( 1) pages  10- 10   16 lines
 7 SimState.java....... sheets   8 to   8 ( 1) pages  11- 11   64 lines
 8 Simulator.java...... sheets   9 to   9 ( 1) pages  12- 13   91 lines
 9 Team.java........... sheets  10 to  10 ( 1) pages  14- 15  122 lines
10 TeamState.java...... sheets  11 to  11 ( 1) pages  16- 16   16 lines
11 MAPAgent.java....... sheets  12 to  12 ( 1) pages  17- 17   34 lines
12 The................. sheets  12 to  12 ( 1) pages  17- 17    1 lines
13 massim.............. sheets  12 to  12 ( 1) pages  17- 17    1 lines
14 package............. sheets  12 to  12 ( 1) pages  17- 17    1 lines
```