ComS 229 – Fall 2010
Project 2 (1000 points)

Introduction

In this project you will write a simulator for a simple robot system and environment. Using the simulator, you will then define a robot control and an environment to perform specified tasks. Optionally you may create a robot that competes with other robots from other students in the course.

Simulation Environment

The simulation environment consists of a grid of squares with length and width to be specified by the configuration. (The configuration is read from a configuration file defined later.) The maximum size of the length or width is 30, and the minimum size is 1. The simulation environment may be rectangular. Each square in the simulation may contain objects and may also have properties.

Objects: Objects may be placed in any empty square in the simulation. Examples of objects are Robot, Rock, Hole, etc. The robot object is special and is described separately below.

Properties: Properties are conditions in the square that may modify how object act that are also in the same square. Some examples of properties are Fog, Standing Water, Darkness, Blinding Light, etc.

Robot Object

The robot object is special and has the following characteristics:

1. The robot object has a front and a back.
2. The robot object has a "battery" that contains positive integer energy units.
3. The robot may turn 90 degrees left or 90 degrees right at an energy cost defined in the configuration file.
4. The robot may move forward one square at an energy cost defined in the configuration file.
5. The robot may probe for "seeable" objects with energy cost and "beam width" defined in the configuration file. The beam width is an odd number that defines the width in number of cells in front of the robot to scan. For example, a beam width of 1 will scan the column directly in front of the robot. A beam width of 3 will scan the three columns directly in front of the robot. The diagram below clarifies how this works. The > is the robot with the point facing forward. The – character represents the beam in the diagram.

```
  ----------
>----------
  ----------
```

Each beam returns the distance to the nearest "seeable" object in its path, or 0 if there is no object in its path. The cost of using a probe is multiplied by the number of beams used to get the total cost to the robot.

6. Two robot objects may never occupy the same square. If they do try to occupy the same square, then the result is dependent on which robot was moving. If one robot is moving and the other robot is not moving or turning, then the moving robot moves into the square and pushes the non-moving or turning robot into the next square. If the next square contains a movable object then all 3 moves. This chain can be repeated with any number of movable objects. However, if any of the objects are not movable, then the original robot that moved remains in the same square. Note that the edge of the simulation area is not movable and objects cannot fall off.

   If the two robots attempt to move into the same square (both robots move) then neither robot moves and they remain in the same square.

7. A robot may fire an energy pulse in the direction it is facing. The cost of the energy pulse is configurable by the formula a+b*e, where e is the amount of energy (strength) of the energy pulse. The parameters a and b are configurable and loaded from the configuration file. If an energy beam with energy e hits another object, it may act on that object in various ways. In the case that it hits another robot object, energy is removed from the hit robot by the formula c*e, where c is loaded from the configuration file. Note that a, b and c are real numbers (not integers) and that each robot has their own set of a, b, and c parameters.

8. A robot always "sees" the contents of the 8 squares surrounding it.

9. A robot may only perform one "action" per turn.

Environment Objects

Other objects in the simulation may be present in the simulation environment. These objects, their attributes, and how they affect the simulation are described below. Note that the simulation may create new environment objects anytime during the simulation, depending on the type of simulation. This is defined under the simulation tag in the configuration file.

1. Rock
   A rock is an immovable object that is placed in the square. A rock will return an echo to the probe and hide any other object behind the rock. Other attributes of rocks are shown in configuration file and have the obvious meanings. Note that immovable applies to the Robot being able to push the rock. This does not preclude the simulation from making rock monsters that move around on their own.

2. Lava
   A lava object represents a pool of lava. Robots may traverse squares with lava pools at an extra energy cost. If the robot does not have the energy to protect it from the lava, the robot is destroyed. Lava is not seen by a probe from a robot.

3. Water
   A water object represents a pool of water. Robots may traverse squares with water pools at an extra energy cost. If the robot does not have the energy to move through the water, it is stuck until it gains enough energy. Water is not seen by a probe from a robot.

4. Mud
   A mud object represents a pool of mud. Robots may traverse squares with mud pools at an extra energy cost. If the robot does not have the energy to move through the mud, it is stuck until it gains enough energy. Even if a robot does have the energy to cross, it is forced to wait some minimum number of turns in the mud. Mud is not seen by a probe from a robot.

5. Ball
   A ball represents a very large ball that robots may push around. Balls may have a specified color, and if pushed by a robot they will move. When pushed by a robot in a direction for n moves, the ball will continue to roll in that direction for an additional n squares, if possible. A ball rolling on its own is stopped by any movable or immovable object. A ball will return an echo to the probe and hide any other object behind the ball.

6. Block
   A block is similar to a ball except that if the robot stops pushing the block, the block immediate stops moving. A block will return an echo to the probe and hide any other object behind the block.

7. Hole
   A hole should be avoided by robots at all costs as anything that moved, or is moved into a hole is removed from the simulation.

8. Energy Pill
   An energy pill object is not movable and if a robot enter the same square as the pill, the energy from the pill is immediately transferred to the robot and the pill is removed from the square. Note that an energy pill may coexist in the same square, with mud, lava, and water objects.

Properties

Properties may exist in combination with any object or other property and affect how robots and objects interact in the square. The following are valid properties.

1. Fog
   The fog property obscures the vision of the robot of the 8 squares surrounding the robot. The only information given the robot is that the square contains fog. No other object or property is reported to the robot. Note that placing the fog property in a square that contains a hole is particularly evil.

2. Jam
   If a square contains the jam property, any robot in the same square that tries to use probes will fail and the energy costs deducted. The jam property is not seeable by a robot.

Initial Configuration File

The initial configuration file specifies initial locations and attributes of objects and properties. The format of this file consists of the specification for each object using XML-like syntax. An example of this syntax is shown below.

```
<Simulation>  // Keywords inside <> are NOT case sensitive
width = 30    // This attribute is required and must be present
height = 10   // This attribute is required and must be present
steps = 300   // This attribute is required and must be present
</simulation>
```

```
<Object>      // Characters after the // are ignored
type = robot  // This attribute is required and must be present
xloc = 4      // This attribute is required and must be present
yloc = 7      // This attribute is required and must be present
color = blue  // May not be present – default value = blue
name = Robot 1// May not be present – default value = Object
display = BR  // This attribute is required and must be present
energy = 200  // May not be present – default value = 100
recharge = 2  // May not be present – default value = 1
movecost = 1  // May not be present – default value = 1
turncost = 1  // May not be present – default value = 1
probecost = 1 // May not be present – default value = 1
parmA = 1.1   // May not be present – default value = 0
parmB = 1.2   // May not be present – default value = 1
parmC = 1.0   // May not be present – default value = 1
</Object>

<Object>
type = rock  // This attribute is required and must be present
xloc = 2     // This attribute is required and must be present
yloc = 2     // This attribute is required and must be present
name = Rock  // May not be present – default value = Object
display = RO // This attribute is required and must be present
</Object>

<Object>
type = lava  // This attribute is required and must be present
xloc = 2     // This attribute is required and must be present
yloc = 4     // This attribute is required and must be present
name = Lava Pit  // May not be present – default value = Object
display = LP // This attribute is required and must be present
damage = 10   // May not be required – default value = 5
</Object>

<Object>
type = water // This attribute is required and must be present
xloc = 2     // This attribute is required and must be present
yloc = 5     // This attribute is required and must be present
name = Water Pool // May not be present – default value = Object
display = WP // This attribute is required and must be present
damage = 1    //  May not be present – default value = 1
</Object>
```

```
<Object>
type = mud   // This attribute is required and must be present
xloc = 3     // This attribute is required and must be present
yloc = 5     // This attribute is required and must be present
name = Mud Swamp // May not be present – default value = Object
display = MS // This attribute is required and must be present
delay = 3    // May not be preset – default value = 2
damage = 5   // May not be present – default value = 2
</Object>

<Object>
type = ball // This attribute is required and must be present
xloc = 8    // This attribute is required and must be present
yloc = 5    // This attribute is required and must be present
color = red // May not be present – default value = blue
name = Big Red Ball // May not be present – default value = Object
display = RB // This attribute is required and must be present
</Object>

<Object>
type = block // This attribute is required and must be present
xloc = 8     // This attribute is required and must be present
yloc = 5     // This attribute is required and must be present
color = blue // May not be present – default value = blue
name = Big Blue Block // May not be present – default value = Object
display = BB // This attribute is required and must be present
</Object>

<Object>
type = hole  // This attribute is required and must be present
xloc = 7     // This attribute is required and must be present
yloc = 5     // This attribute is required and must be present
name = A Little Hole // May not be present – default value = Object
display = H  // This attribute is required and must be present
</Object>

<Object>
type = energy // This attribute is required and must be present
xloc = 6      // This attribute is required and must be present
yloc = 5      // This attribute is required and must be present
name = Energy Pill // May not be present – default value = Object
display = EP  // This attribute is required and must be present
</Object>
```

```
<Property>
type = fog    // This attribute is required and must be present
xloc = 6      // This attribute is required and must be present
yloc = 6      // This attribute is required and must be present
name = Fog    // May not be present – default value = Property
display = F   // This attribute is required and must be present
</Property>

<Property>
type = jam    // This attribute is required and must be present
xloc = 6      // This attribute is required and must be present
yloc = 8      // This attribute is required and must be present
name = Jam    // May not be present – default value = Property
display = J   // This attribute is required and must be present
</Property>
```

Part a (250 points) Write a program called "checkconf" that takes as a parameter in the command line the name of the file that contains configuration data and processes the contents as follows. The format of the configuration file will be as defined above. The program reads and parses the contents of the file and then outputs a representation of the simulation environment in text format along with a sorted list of all the object sorted by location. (Row number used first to determine order with ties broken by the column number.) An example output for a small simulation environment is shown below. For each square the first to characters are the first two characters found in the display attribute. (If there is only a single character then the second character is blank.) The third character shown for the square represents a property. If a square has more than one property, then an X appears in the this character position. Otherwise, only the first character of the attribute for the property is displayed. If there are two objects located in the same square, then "XX" is displayed.

```
... ... ... RO. ... ...
... ... ... ... ... ...
... LPF ... ... RO. ...
... BR. ..F ... BB. ...

Location 0, 3
Rock

Location 2, 1
Lava Pit
Fog
```

```
Location 2, 4
Rock

Location 3, 1
Robot 1
Energy 200
Color blue  // Do not display the other parameters of the robot

Location 3, 2
Fog

Location 3, 4
Big Blue Block
Color blue
```

Part b (350 points) Write a simulator programed called "cs229sim" that reads a configuration file and simulates the world according to the rules defined above. Because robots may make decisions, a program is used to direct their movements. For this reason, robots must subclass the IRobot class and implement the methods. Note that your simulation will need to create supporting objects. The ISimObject represents any object or property in the simulation. The ISee object represents what a robot sees in the surrounding 8 squares at any given time. Sqaure 0 is directly in front of the robot, square 2 is directly to the right, moving clockwise around the robot for all 8 squares. It contains an array of Lists as explained on page 166 of Josuttis. The IProbe object represents any probing performed from the last turn. This also returns a list since the probe may be multiple beams wide. Probe 0 is always the beam directly in front of the robot. Beam 1 is directly to the left, beam to is directly to the right, beam 3 is second to the left, etc.

For the IMove interface the simulation must follow the following convention.

Action = 0: Do nothing.
Action = 1: Move forward.
Action = 2: Turn right
Action = 3: Turn left
Action = 4: Probe, Parameter indicates how many beams to use.
Action = 5: Fire energy pulse, Parameter indicates how much energy to use.

```
#include <cstdlib>
#include <string>
#include <list>

using namespace std;

class ISimObject
{
  public:
    string getName()=0;
};


class ISee
{
  list<ISimObject> look( int x ) = 0;
};

class IProbe
{
  List<ISimObject> probe() = 0;
};
```

```
class IMove
{
  public:
    int getAction() = 0;
    int getParameter() = 0;
};

class IRobot
{
  public:
    IMove getMove( ISee s, IProbe p) = 0;
};
```

In addition you must also utilize the ISimulation class that contains the following methods.

```
class ISimulation
{
  public:
    void loadConfiguration( string file) = 0;
    void setRobot( int index, IRobot) = 0;
    void step( int s ) = 0;
};
```

The loadConfiguration loads a simulation from a configuration file. The setRobot method sets the IRobot object for each of the robots defined in the configuration file. Index 0 is the first robot defined, index 1 is the second, etc. The step method performs the simulation for a specified number of steps.

Part c (250 points) Write a robot that performs tasks.

Write a program called "create" that automatically creates a configuration file with the following specifications.

- The simulation size is 30 wide and 10 high.
- The simulation time is 300 moves.
- The simulation area is divided into three zones. The first zone (the play zone) is the middle 20 columns where all the interesting things happen. The two zones 5 wide at the edges are safe zones.
- Randomly place 3 holes, 10 rocks, 1 lava, 3 water, 3 mud in the play zone.
- Randomly place 5 fog and 4 jam properties in the play zone.
- Randomly place 2 energy pills in the play zone.
- Randomly place 3 red balls and 3 red blocks in the left safe zone.

- Randomly place 3 blue balls and 3 blue blocks in the right safe zone.
- Randomly place 1 red robots in the right safe zone.
- Randomly place 1 blue robots in the left safe zone.

The object of the game is for the red robot1 to get all of its colored balls in the right safe zone, and for the blue robot to get all of its colored balls in the left safe zone. The game ends when either all the objects of the right color are in the target safe zone, or the simulation time runs out. If this happens, the robot with the most colored objects in the correct safe zone wins.

Part d (50 points) Correctly working makefile.

Part e (100 points) OO Design document.