Assignment 4

SFWR ENG 2AA4

Specification due Mar 11, Code submission due Mar 18

The purpose of this software design exercise is to design and implement a portion of the specification for an autonomous rescue robot. You are given a partial specification and asked to fill in the specification of the missing semantics. Once the specification is complete, you will implement a portion of the version provided by the instructor.

The motivation for the current problem is the capstone design project from last year (2006–2007). In this project teams of 5 or 6 students developed Remote Image Guided Autonomous Rescue Robots (RIGARR). The inspiration for the project is real life rescue robots, which are used when a disaster occurs and the conditions are too dangerous for human rescuers. To determine the path for the rescue mission the teams were given a digital image showing the destinations they had to reach and the obstacles blocking their path. Teams competed to reach all of the destinations in the shortest time. The hardware used for robot construction was the Lego Mindstorms NXT kit.

The focus of the current assignment will only be on the route planning portion of the above project. The information on obstacles and destinations will be assumed to be available to the modules designed in this exercise. The modules for this assignment deal with determining a path from a safe zone back to the safe zone, while passing through all rescue regions and avoiding all obstacles. Figure **??** shows a map of the area of interest. The lower left corner of the map is located at the origin of the x-y coordinate system. The map has length MAX_X in the x direction and length MAX_Y in the y direction. Any point outside of the map area is considered to be an invalid point. Within the map there are rectangles (regions) for the safe zone, the rescue regions (destinations), and for the obstacles. Each rectangle is defined by the coordinate of its lower left corner, together with values for its width and its height. This information is identified on Figure **??** for one of the obstacle regions.

To rescue all of the potential victims, a valid path proceeds from the safe zone back to the safe zone, visits all of the rescue regions, does not cross any of the obstacles and respects that stated tolerances. The purpose of the tolerances is to allow for the fact that the robot may not be exactly where you plan it to be. The robot cannot be closer

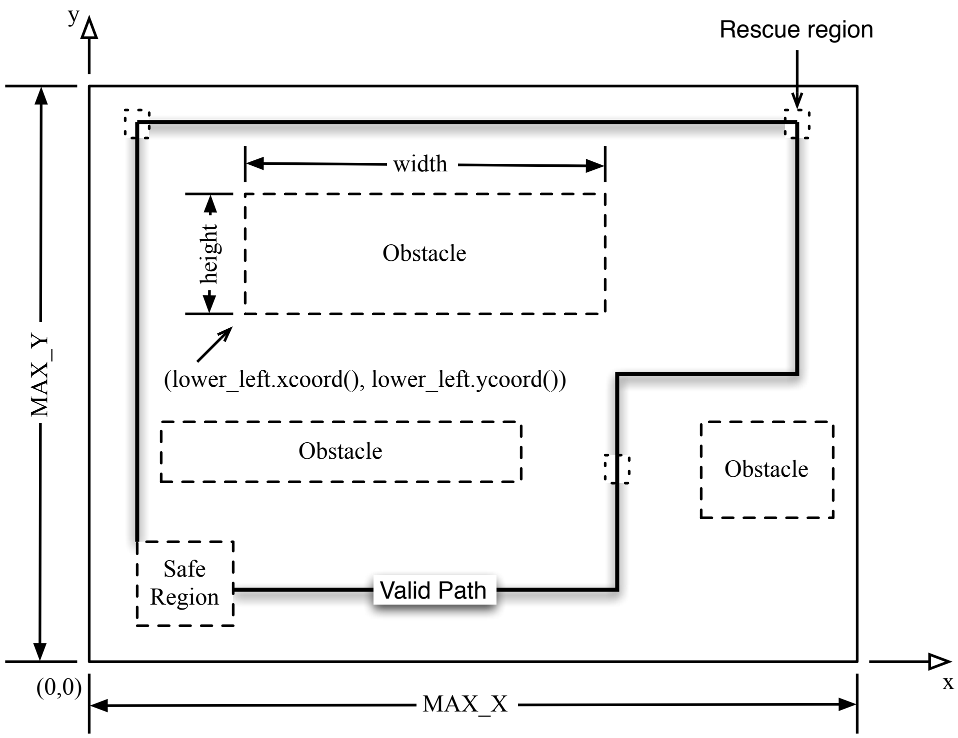

Figure 1: Example map with valid and invalid paths

than TOLERANCE to an obstacle to take this into account. The robot is also allowed to "miss" the rescue regions and the safezone by the TOLERANCE amount. A path is made up of a sequence of points, where each point is defined as a tuple of x and y coordinates. The path is defined as the straight line connecting subsequent points.

The modules specified at the end of this assignment description are as follows: Constants, PointT, RegionT, GenericList(T), PathT, Obstacles, Destinations, SafeZone, Map and PathCalculation. A portion of the specification is given, but within it there are several mathematical specifications that you need to design. Your specifications should not involve writing algorithms or pseudo-code. The specifications should use discrete mathematics to specify the desired properties. That is, you should be writing a *descriptive* specification as opposed to an *operational* specification. Specifications within a module are free to use access programs defined within the current module or from another module that is used by the current module.

Step 1

Complete the specification for the RegionT module. You will need to complete the following:

RegionT(p, w, l): Write the mathematical specification for the InvalidRegionException exception in the constructor for RegionT. This exception should be thrown when any portion of input region would extend outside of the map area, as defined in Figure **??**.

pointInRegion(p): Write the output portion of the specification. This routine should return true if the point p is within TOLERANCE of the region. That is, if the distance from the point p to any point within the region is less than TOLERANCE, then return true.

Step 2

Complete the specification of the semantics portion of the PathCalculation module. A description of the behaviour of each of the access programs is as follows:

is_validSegment(p_1, p_2): This routine should return true if the line segment between p_1 and p_2 does not come any closer than TOLERANCE to any of the obstacles.

is_validPath(p): This routine returns true if the path is valid. A valid path must begin and end within TOLERANCE of the safe zone region. The path must pass within

TOLERANCE of all of the rescue regions and none of the points in the line segments connecting subsequent points in the path should come closer than TOLERANCE to any of the obstacles.

is_shortestPath(p): This routine returns true if the path p is the shortest of all valid paths.

totalDistance(p): This routine returns the sum of the lengths of the piecewise segments that make up the sequence of points in the path.

totalTurns(p): This routine returns the number of turns in the path p. A turn is any change of the orientation of the robot.

estimatedTime(p): This routine returns the estimated time for traversing the path p. The time is calculated as the sum of the times to traverse the straight segments and the times to do all of the turns. The time for covering a straight segment is calculated using Constants.VELOCITY_LINEAR. The time for turning is calculated using the angle of the turn (in radians) and Constants.VELOCITY_ANGULAR. You may assume that all constants are in

Step 3

Submit a report showing the specifications from the previous steps. The specifications can be hand-written and you do not need to reproduce the portions of the specification that have already been provided to you. Your report should be accompanied by the portion of your logbook relevant to this assignment.

Step 4

After the report has been submitted, you will be provided with a complete specification for all of the modules. Implement the modules in Java. The names of the modules that need to be implemented are as follows: `Constants.java`, `PointT.java`, `InvalidPointException.java`, `RegionT.java`, `InvalidRegionException.java`, `PathT.java`, `Obstacles.java`, `Destinations.java`, `SafeZone.java`, `FullSequenceException.java`, `InvalidPositionException.java`, `Map.java` and `PathCalculation.java`. For the `PathCalculation.java` class, you do not need to implement the methods for `is_validSegment`, `is_validPath` and `is_shortestPath`.

Step 5

Write a fourth module, names `TestPathCalculation.java` that tests the implemented routines of the PathCalculation module. You can test other routines as well, but you are only required to test the PathCalculation routines. Each procedure should have at least one test case. For this assignment you are not required to submit a lab report, but you should still carefully think about your rationale for test case selection. Please make an effort to test normal cases, boundary cases, and exception cases. Your test program should have the test cases "hard coded" into the program, rather than expecting user input. Your test program should also automatically compare the calculated output to the expected output and automatically state whether the test case has passed or not. Your test program should keep the following counts: tests cases passed, test cases failed, and total number of test cases. The end of your test program should summarize these counts.

Step 6

Submit all Java files to the subversion repository. This must be completed no later than midnight of the deadline for file submission.

Notes

1. Please put your name and student number at the top of each of your source files.
2. Your program must work in the ITB labs on moore when compiled by javac and run using java.
3. Please use `double` in your implementation of real.
4. Your Subversion submission should be in the folder `Assig4`.
5. All exceptions should be RunTimeExceptions and they should have a constructor that takes a string argument. The string provided when the exception is thrown will be an explanation of the error.
6. The robot is assumed to only move forward, so the specification does not need to worry about a robot that can drive backwards.
7. The grading for this assignment will be 50 % for part 1 (the specification) and 50 % for part 2 (the implementation)

8. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**

Constants Module

Module

Constants

Uses

N/A

Syntax

Exported Constants

MAX_X = 180 *//dimension in the x-direction of the problem area*
MAX_Y = 160 *//dimension in the y-direction of the problem area*
TOLERANCE = 5 *//space allowance around obstacles*
VELOCITY_LINEAR = 15 *//speed of the robot when driving straight*
VELOCITY_ANGULAR = 30 *//speed of the robot when turing*

Exported Access Programs

none

Semantics

State Variables

none

State Invariant

none

Point ADT Module

Template Module

PointT

Uses

Constants

Syntax

Exported Types

PointT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
PointT	real, real	PointT	InvalidPointException
xcoord		real	
ycoord		real	
dist	PointT	real	

Semantics

State Variables

xc: real
yc: real

State Invariant

none

Assumptions

The constructor PointT is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

PointT(x, y):

- transition: $xc, yc := x, y$
- output: $out := self$
- exception

  $exc := ((\neg(0 \leq x \leq \text{Contants.MAX\_X}) \vee \neg(0 \leq y \leq \text{Constants.MAX\_Y})) \Rightarrow \text{InvalidPointException})$

xcoord():

- output: $out := xc$
- exception: none

ycoord():

- output: $out := yc$
- exception: none

dist(p):

- output: $out := \sqrt{(self.xc - p.xc)^2 + (self.yc - p.yc)^2}$
- exception: none

Region Module

Template Module

RegionT

Uses

PointT, Constants

Syntax

Exported Types

RegionT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
RegionT	PointT, real, real	RegionT	InvalidRegionException
pointInRegion	PointT	boolean	

Semantics

State Variables

lower_left: PointT *//coordinates of the lower left corner of the region*
width: real *//width of the rectangular region*
height: real *//height of the rectangular region*

State Invariant

None

Assumptions

The RegionT constructor is called for each abstract object before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

RegionT(p, w, h):

- transition: $lower\_left, width, height := p, w, h$
- output: $out := self$
- exception: $exc :=?$

pointInRegion(p):

- output: $out :=?$
- exception: none

Generic List Module

Generic Template Module

GenericList(T)

Uses

N/A

Syntax

Exported Types

GenericList(T) = ?

Exported Constants

MAX_SIZE = 100

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
GenericList		GenericList	
add	integer, T		FullSequenceException, InvalidPositionException
del	integer		InvalidPositionException
setval	integer, T		InvalidPositionException
getval	integer	T	InvalidPositionException
size		integer	

Semantics

State Variables

s: sequence of T

State Invariant

$|s| \leq$ MAX_SIZE

Assumptions

The GenericList() constructor is called for each abstract object before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

GenericList():

- transition: $\mathit{self}.s := <>$
- output: $\mathit{out} := \mathit{self}$
- exception: none

add(i, p):

- transition: $s := s[0..i-1] || <p> || s[i..|s|-1]$
- exception: $\mathit{exc} := (|s| = \text{MAX\_SIZE} \Rightarrow \text{FullSequenceException} \mid i \notin [0..|s|] \Rightarrow \text{InvalidPositionException})$

del(i):

- transition: $s := s[0..i-1] || s[i+1..|s|-1]$
- exception: $\mathit{exc} := (i \notin [0..|s|-1] \Rightarrow \text{InvalidPositionException})$

setval(i, p):

- transition: $s[i] := p$
- exception: $\mathit{exc} := (i \notin [0..|s|-1] \Rightarrow \text{InvalidPositionException})$

getval(i):

- output: $\mathit{out} := s[i]$
- exception: $\mathit{exc} := (i \notin [0..|s|-1] \Rightarrow \text{InvalidPositionException})$

size():

- output: $\mathit{out} := |s|$
- exception: none

Path Module

Template Module

PathT is GenericList(PointT)

Obstacles Module

Template Module

Obstacles is GenericList(RegionT)

Destinations Module

Template Module

Destinations is GenericList(RegionT)

SafeZone Module

Template Module

SafeZone extends GenericList(RegionT)

Exported Constants

MAX_SIZE = 1

Map Module

Module

Map

Uses

Obstacles, Destinations, SafeZone

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
init	Obstacles, Destinations, SafeZone		
get_obstacles		Obstacles	
get_destinations		Destinations	
get_safeZone		SafeZone	

Semantics

State Variables

$obstacles$: Obstacles
$destinations$: Destinations
$safeZone$: SafeZone

State Invariant

none

Assumptions

The access routine init() is called for the abstract object before any other access routine is called. If the map is changed, init() can be called again to change the map.

Access Routine Semantics

init(o, d, sz):

- transition: $obstacles, destinations, safeZone := o, d, sz$

- exception: none

get_obstacles():

- output: $out := obstacles$
- exception: none

get_destinations():

- output: $out := destinations$
- exception: none

get_safeZone():

- output: $out := safeZone$
- exception: none

Path Calculation Module

Module

PathCalculation

Uses

Constants, PointT, RegionT, PathT, Obstacles, Destinations, SafeZone, Map

Syntax

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
is_validSegment	PointT, PointT	boolean	
is_validPath	PathT	boolean	
is_shortestPath	PathT	boolean	
totalDistance	PathT	real	
totalTurns	PathT	integer	
estimatedTime	PathT	real	

Semantics

?