Table Of Content

Class Compass

```
java.lang.Object
    |
    +--Compass
```

< Constructors > < Methods >

public class **Compass**
extends java.lang.Object

Constructors

Compass

```
public  Compass(Robot.Direction head,
                Robot.Direction robot)
```

Compass constructor.

Parameters:

head - Initial orientation of robot's head.
robot - Initial orientation of robot.

Methods

getDirection

```
public Robot.Direction getDirection()
```

Returns a Robot.Direction that corresponds to the direction the robot is facing.

Returns:

the direction the robot is facing

getTurnDegrees

```
public int getTurnDegrees(Robot.Direction direction)
```

Takes in the direction that you want the robot to be facing and returns how many degrees need to be passed to DifferentialPilot.rotate() to achieve that orientation.

Parameters:

direction - The direction you want the robot to face.

Returns:

Degrees required to achieve that direction.

---

gethead

```
public Robot.Direction gethead()
```

Returns a Robot.Direction that corresponds to the direction the robot's head is facing.

Returns:

the direction the robot's head is facing

---

robotTurned

```
public Robot.Direction robotTurned(Robot.Direction direction)
```

Takes in the direction that the robot has turned and updates the compass' record of which direction the robot is facing. eg. if the robot is facing left and it turns right. It will be facing forwards.

Parameters:

direction - The direction the robot turned

Returns:

The Robot's new direction.

---

turnHeadDegrees

```
public int turnHeadDegrees(Robot.Direction direction)
```

Takes in the direction that you want the robot's head to be facing and returns how many degrees the head motor needs to turn to face in that direction.

Parameters:

direction - The direction you want the robot's head to face.

Returns:

Degrees required to achieve that direction.

Class Robot

```
java.lang.Object
    |
    +--Robot
```

< Constructors > < Methods >

public class **Robot**
extends java.lang.Object

Constructors

Robot

`public  Robot(WorldMap map)`

Methods

bumperPressed

`public boolean bumperPressed()`

Returns whether the either bumper has been pressed.

Returns:

Whether the bumper has been pressed or not.

checkDirectionOccupied

`public boolean checkDirectionOccupied(Robot.Direction direction)`

Takes in a cardinal direction and returns whether the cell in that direction is occupied or not.

Parameters:

direction - Direction to check.

Returns:

Whether the direction is occupied or not.

checkSurroundings

`public void checkSurroundings()`

Turns the robot's head and checks front, left and right of robot for obstacles. Records what it finds in the world map. This includes whether the cells it looks into are unoccupied. Range checked: = 25cm

---

chooseACell

`public Robot.Direction chooseACell()`

Movement algorithm. Called every time the robot moves to choose it's new direction to head. Splits the arena into four quadrants. It then works out the percentage of each quadrant that is unvisited. Each cardinal direction (north, south, east and west) is then given a priority based upon the two quadrants that contain cells in it's direction. I.E. North gets the priorities of northwest and northeast. South gets the priorities of southwest and southeast. East gets the priorities of northeast and southeast. West gets the priorities of northwest and southwest. The cardinal direction with the highest priority from these additions is then selected as the direction to head in. Obstacles are taken into account. A higher priority direction will be ignored if there is an obstacle. Problem: This algorithm suffers the same problem as the system that functions like a gradient, where the robot is repelled by obstacles and drawn by the goal. It will get stuck down alleys. This was not seen as a problem due to the scattered nature of the arena's obstacles.

Returns:

Robot.Direction chosen to head in.

---

getCurrentColumn

`public int getCurrentColumn()`

Robot's current column getter.

Returns:

Current column of robot.

---

getCurrentRow

`public int getCurrentRow()`

Robot's current row getter.

Returns:

Current row of robot.

---

getMap

```
public WorldMap getMap()
```

WorldMap getter.

Returns:

World Map that robot is using.

---

moveForwards

```
public void moveForwards()
```

Moves the robot forwards one cell. Also updates the robot's current location. Also updates the world map's occupancy and visit counts to reflect that the cell entered has been examined.

---

returnGreatest

```
public int returnGreatest(double[] priorities)
```

Gets the index of the highest value in an array of priorities.

Parameters:

priorities - An array of priorities.

Returns:

The index of the highest priority.

---

setCurrentColumn

```
public void setCurrentColumn(int column)
```

Sets robot's current column. Does not allow robot location to be set outside arena.

Parameters:

column - Robot's new column.

---

setCurrentLocation

```
public void setCurrentLocation(int column,
                               int row)
```

Sets robot's current column and row. Does not allow robot location to be set outside arena.

Parameters:

column - Robot's new column.
row - Robot's new row.

setCurrentRow

```
public void setCurrentRow(int row)
```

Sets robot's current row. Does not allow robot location to be set outside arena.

Parameters:

row - Robot's new row.

turnHead

```
public void turnHead(Robot.Direction direction)
```

Turns the head to face the direction given.

Parameters:

direction - The direction the head needs to turn to face, relative to the robot's body.

turnRobotToFace

```
public void turnRobotToFace(Robot.Direction directionChosen)
```

Turns robot to face cardinal direction passed to it. Updates compass to keep track of new direction.

Parameters:

directionChosen - Direction robot should face.

Class Robot.Direction

```
java.lang.Object
   |
   +--java.lang.Enum
         |
         +--Robot.Direction
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

< Fields > < Methods >

public static final class **Robot.Direction**
extends java.lang.Enum

Fields

BACKWARDS

`public static final Robot.Direction **BACKWARDS**`

FORWARDS

`public static final Robot.Direction **FORWARDS**`

LEFT

`public static final Robot.Direction **LEFT**`

RIGHT

`public static final Robot.Direction **RIGHT**`

Methods

valueOf

`public static Robot.Direction **valueOf**(java.lang.String name)`

---

values

`public static Robot.Direction[] **values**()`

Class WorldMap

```
java.lang.Object
    |
    +--WorldMap
```

< Constructors > < Methods >

---

public class **WorldMap**
extends java.lang.Object

Constructors

WorldMap

```
public  WorldMap(int arenaLength,
                 int arenaWidth,
                 int cellLength,
                 int cellWidth)
```

Constructor for world map. cellLength and cellWidth should always be 25 for now as variable movement distances are unimplemented.

Parameters:

arenaLength - Length of the arena
arenaWidth - Width of the arena
cellLength - Length of individual cells.
cellWidth - Width of individual cells.

Methods

cellOccupied

```
public void cellOccupied(int x,
                         int y)
```

Increments a cell's occupancy count by 1. Also increments the cells visit count.

Parameters:

x - X co-ordinate.
y - Y co-ordinate.

---

cellUnoccupied

```
public void cellUnoccupied(int x,
                           int y)
```

Decrements a cell's occupancy count by 1. Also increments the cells visit count.

Parameters:

x - X co-ordinate.
y - Y co-ordinate.

---

fullyExplored

```
public boolean fullyExplored()
```

Checks whther the entire map has been explored or not.

Returns:

Whether the map has been explored or not.

---

getColumnLength

```
public int getColumnLength()
```

columns getter.

Returns:

How many columns the world has.

---

getOccupancyCount

```
public int getOccupancyCount(int x,
                             int y)
```

Occupancy count getter for individual cells

Parameters:

x - Column to get occupancy count from.
y - Row to get occupancy count from.

Returns:

The occupancy count of the cell.

---

getOccupancyProbabilityCell

```
public float getOccupancyProbabilityCell(int column,
                                         int row)
```

Gets the occupancy probability of a single cell. Returns 1 (100% chance of occupancy) for out of bounds cells. Returns -1 for unvisited cells.

Parameters:

column - X co-ordinate.
row - Y co-ordinate.

Returns:

Probability the given cell is occupied.

---

getOccupancyProbabilityGrid

```
public float[][] getOccupancyProbabilityGrid()
```

Returns a 2d array of floats that corespond to the world map's occupancy probabilities.

Returns:

2d array of occupancy probabilities.

---

getRowLength

```
public int getRowLength()
```

rows getter.

Returns:

How many rows the world has.

---

getSectionSize

```
public double getSectionSize(int xStart,
                             int xEnd,
                             int yStart,
                             int yEnd)
```

Returns the size of a section

Parameters:

xStart - Right boundary of section.
xEnd - Left boundary of section.
yStart - Bottom boundary of section.
yEnd - Top boundary of section.

Returns:

Total cells in a section.

---

getSectionUnvisitedPercentage

```
public double getSectionUnvisitedPercentage(int xStart,
                                            int xEnd,
                                            int yStart,
                                            int yEnd)
```

Returns the percentage of a given section of the map that has been visited.

Parameters:

xStart - Right boundary of section.
xEnd - Left boundary of section.
yStart - Bottom boundary of section.
yEnd - Top boundary of section.

Returns:

Percentage of the section that has been visited.

---

getVisitCount

```
public int getVisitCount(int x,
                         int y)
```

Visit count getter for individual cells

Parameters:

x - Column to get visit count from.
y - Row to get visit count from.

Returns:

How many times the cell has been visited.

---

isCellOccupied

```
public boolean isCellOccupied(int x,
                              int y)
```

Checks whether a cell is thought to be occupied or not. Returns true if the probability is 50% or higher.

Parameters:

x - Column to check.
y - Row to check.

Returns:

Whether the cell is occupied or not.

---

isCellVisited

```
public boolean isCellVisited(int x,
                             int y)
```

Checks whether a cell has been visited or not.

Parameters:

x - Column to check.
y - Row to check.

Returns:

Whether the cell has been visited or not.

---

outOfBounds

```
public boolean outOfBounds(Robot.Direction direction,
                           int x,
                           int y)
```

Checks whether a cell in a given cardinal direction from a co-ordinate is out of bounds..

Parameters:

direction - Cardinal direction to check in.
x - X co-ordinate.
y - Y co-ordinate.

Returns:

Whether the cell is out of bounds or not.

---

unvisited

```
public double unvisited(int xStart,
                        int xEnd,
                        int yStart,
                        int yEnd)
```

Returns the count of unvisited cells in a section

Parameters:

xStart - Right boundary of section.
xEnd - Left boundary of section.
yStart - Bottom boundary of section.
yEnd - Top boundary of section.

Returns:

Total unvisited cells in the section.

---

visitCell

```
public void visitCell(int x,
                      int y)
```

Marks a cell as having been visited once more.

Parameters:

x - Column to update visit count of.
y - Row to update visit count of.

INDEX