Simulation Code Writeup
First Quarter 2009-2010

Jeff Hobson

April 9, 2010

1 Environment

1.1 Code

```
File: ALMSim.pde
final int WIDTH     = 400;
final int HEIGHT    = WIDTH+5;
final int MARGIN    = 50;
boolean paused      = false;
int[] stdGrassColor = { 50, 160, 50};
int[] cutGrassColor = { 50, 255, 50};
int[] groundColor   = {115,  50,  0};
int[] obstacleColor = {255,  75, 75};
double[][] floorType   = new double[HEIGHT][WIDTH]; // 0 = uncut; 1 = cut; 2 = gnd; 3 =
Robot ALM;
void setup()
{
  size(500,500);
  stroke(0);
  fill(255);
  ALM = new Robot();
  //hard-coded or randomly-generated map
  //cut
  for(int r=25; r<75; r++)
    for(int c=25; c<75; c++)
      floorType[r][c] = 1;
  //ground
  //for(int r=100; r<125; r++)
  //  for(int c=25; c<75; c++)
  //    floorType[r][c] = 2;
  //obstacle(s)
  /*for(int r=150; r<175; r++)
    for(int c=25; c<75; c++)
```

```
      floorType[r][c] = 3;*/
  //noLoop();
  //startSim();
}

void draw()
{
  //grow grass
  for(int e=0; e<HEIGHT; e++)
    for(int f=0; f<WIDTH; f++)
      if(floorType[e][f]>0 && floorType[e][f]<=1)
        floorType[e][f] -= .001;
  //cut
  ALM.cutForward();
  //borders
  noFill();
  stroke(0);
  rectMode(LEFT);
  rect(MARGIN-1,MARGIN-1,WIDTH+MARGIN,HEIGHT+MARGIN);
  //map
  for(int r=0; r<HEIGHT; r++)
  {
    for(int c=0; c<WIDTH; c++)
    {
      if(floorType[r][c] <= 0)
      {
        stroke(stdGrassColor[0], stdGrassColor[1], stdGrassColor[2]);
        point(MARGIN+c,MARGIN+r);
      }
      else if(floorType[r][c] == 1)
      {
        stroke(cutGrassColor[0], cutGrassColor[1], cutGrassColor[2]);
        point(MARGIN+c,MARGIN+r);
      }
      else if(floorType[r][c] == 2)
      {
        stroke(groundColor[0], groundColor[1], groundColor[2]);
        point(MARGIN+c,MARGIN+r);
      }
      else if(floorType[r][c] == 3)
      {
        stroke(obstacleColor[0], obstacleColor[1], obstacleColor[2]);
        point(MARGIN+c,MARGIN+r);
      }
      /*else if(floorType[r][c] == 4)
      {
        int[] co = ALM.getColor();
        stroke(co[0],co[1],co[2]);
        point(MARGIN+c,MARGIN+r);
      }*/
      else
      {
        stroke(50,160+(int)(95*floorType[r][c]),50);
        point(MARGIN+c,MARGIN+r);
      }
    }
  //robot
  ALM.Draw();
  }
}

void mouseClicked()
{
  if(paused)
    loop();
  else
    noLoop();
  paused = !paused;
}

void startSim()
{
  int count = 0;
  while(count<(WIDTH-5)/ALM.getSpeed())
  {
    ALM.cutForward();
    redraw();
    count++;
  }
}
```

1.2 Discussion

This code is creates the environment for the robot (the lawn). It is created manually (hard-coded) for now, but it will eventually have the capability to create random lawns with obstacles of different sizes and shapes. The ALMSim class instantiates the Robot object in the `setup()` method with the default values. The `draw()` method is a method that is required in every Processing base class and is looped indefinitely by the program until the user decides to exit. This method will update the lawn based on the value of each pixel and also calls the Robot's `Draw()` method, which prints the lawnmower to the screen.

2 Object

2.1 Code

2.1.1 Overlap ALM

```
File: ALMSim_overlap/Robot.pde
class Robot
{
  private final double left      =  90;
  private final double right     = 270;
  private final double halfLeft  =  45;
  private final double halfRight = 225;
  private int row, col;   //current location on grid
  private int speed;      //pixels per second
  private int overlap;    //a counter used when determining how far to overlap before tu
  private int roboSize;   //size in pixels
  private int viewRange;  //distance sensors can reach in pixels
  private double heading; //direction in polar grid
  private String type;    //circular or rectangular size
  private int[] roboColor = {
    0,0,220  };

  //Constructors
  public Robot(int r, int c, int sp, double h, int sz, int vr, String t, int[] cl)
  {
    row       =  r;
    col       =  c;
    speed     = sp;
    heading   =  h;
    roboSize  = sz;
    viewRange = vr;
    type      =  t;
    roboColor = cl;
  }
  public Robot(int r, int c, int sp, double h, int sz, int vr)
  {
    row       =  r;
    col       =  c;
    speed     = sp;
    heading   =  h;
    roboSize  = sz;
    viewRange = vr;
    type  = "rect";
  }
  public Robot(int r, int c, int sp, double h)
```

```
{
  row       =  r;
  col       =  c;
  speed     = sp;
  heading   =  h;
  roboSize  =  5;
  viewRange =  3;
  type  = "rect";
}
public Robot()
{
  row       = 350;
  col       = 5;
  speed     = 100;
  heading   = 3.0;
  roboSize  = 5;
  viewRange = 3;
  type = "rect";
}

//Accessors
int[] getPos()
{
  int[] ret = {
    row, col    };
  return ret;
}
int getSpeed()
{
  return speed;
}
double getHeading()
{
  return heading;
}
int getSize()
{
  return roboSize;
}
int getRange()
{
  return viewRange;
}
String getType()
{
  return type;
```

```
}
int[] getColor()
{
  return roboColor;
}

//"Setters"
void setPos(int r, int c)
{
  row = r;
  col = c;
}
void setSpeed(int sp)
{
  speed = sp;
}
void setHeading(int h)
{
  heading = h;
}
void setSize(int sz)
{
  roboSize = sz;
}
void setRange(int vr)
{
  viewRange = vr;
}
void setType(String t)
{
  type = t;
}
void setColor(int[] c)
{
  roboColor = c;
}

//Instance Methods
void Draw()
{
  /*for(int r=-roboSize; r<roboSize; r++)
   for(int c=-roboSize; c<roboSize; c++)
   if(row+r<0 || col+c<0 || row+r>=HEIGHT || col+c>=WIDTH)
   continue;
   else
   floorType[row+r][col+c] = 4;*/
  rectMode(CENTER);
  stroke(255,0,0);
  //stroke(roboColor[0],roboColor[1],roboColor[2]);
  fill(roboColor[0],roboColor[1],roboColor[2]);
  // find four corner points
  beginShape();
  for(double theta=.5; theta<=3.5; theta++)
  {
    vertex(col+MARGIN+roboSize*sqrt(2.0)*cos((float)(heading+theta)*PI/2.0),row+MARGIN+
  }
  endShape(CLOSE);
  //rect(col+MARGIN,row+MARGIN,roboSize*2,roboSize*2);
  stroke(255);
  fill(255);
  ellipse(col+MARGIN+roboSize*cos((float)heading*PI/2.0),row+MARGIN+roboSize*-sin((flo
  //noFill();
}

void cutForward()
{
  //check next line of pixels only.
  //vector -> check 3 points
  boolean flag = false;
  for(int x=0; x<speed; x++)
  {
    // Each step I want to:
    // (1) Get the floor values in front of me
    // (2) Turn if they are obstacles
    // (3) Cut the grass and move one pixel if they are not.
    int[] objsInFront = checkForward(); // returns -1,0,1 relating to which pixel find
    if(objsInFront != null)
    {
      if(objsInFront[1] == 3 || objsInFront[1] == 1)
      {
        if(objsInFront[1] == 1)
        {
          while(overlap<roboSize)
          {
            int R = roboSize;
            for(int r=-R; r<=R; r++)
              for(int c=-R; c<=R; c++)
                if(c*c+r*r<=R*R)
                  floorType[row+r][col+c] = 1;
            int[] pos = forward(col,row);
            col = pos[0];
            row = pos[1];
            overlap++;
          }
          overlap = 0;
        }
        turn(left);
        println(heading);
        break;
      }
      else if(objsInFront[0] == 3)
      {
        turn(halfRight);
        break;
      }
      else if(objsInFront[2] == 3)
      {
        turn(halfLeft);
        break;
      }
      else
      {
        int R = roboSize;
        for(int r=-R; r<=R; r++)
          for(int c=-R; c<=R; c++)
            if(c*c+r*r<=R*R)
              floorType[row+r][col+c] = 1;
        int[] pos = forward(col,row);
        col = pos[0];
        row = pos[1];
        overlap = 0;
      }
    }
  }
  //draw robot again
  //this.Draw();
}
int[] checkForward()
{
  //shit time to change all this
  int[] x = new int[3];
  int[] y = new int[3];
  int R = roboSize;
  //front to the left
  x[0] = col+(int)(R*sqrt(2)*cos((float)(heading+.5)*PI/2.0));
  y[0] = row+(int)(R*sqrt(2)*-sin((float)(heading+.5)*PI/2.0));
  //directly in front
  x[1] = col+(int)(R*cos((float)heading*PI/2.0));
  y[1] = row+(int)(R*-sin((float)heading*PI/2.0));
  //front to the right
  x[2] = col+(int)(R*sqrt(2)*cos((float)(heading-.5)*PI/2.0));
  y[2] = row+(int)(R*sqrt(2)*-sin((float)(heading-.5)*PI/2.0));

  //Check the above three points 1 pixel in front of the robot to see if it is clear o
  int[] retval = new int[3];
  for(int i=-1; i<2; i++)
  {
    x[i+1] += 2*cos((float)heading*PI/2.0);
    println(2*cos((float)heading*PI/2.0));
    y[i+1] -= 2*sin((float)heading*PI/2.0);
    if(x[i+1]<0 || x[i+1]>=WIDTH || y[i+1]<0 || y[i+1]>=HEIGHT) // Treat boundaries as
      retval[i+1] = 3;//i;
    else
      retval[i+1] = floorType[y[i+1]][x[i+1]];
    /*if(floorType[y[i+1]][x[i+1]] == 3 || floorType[y[i+1]][x[i+1]] == 1) // If there
     return i;*/
  }
  return retval;
}
int[] forward(int cola, int rowb)
{
  int c = cola;
  int r = rowb;
  if(heading == 0)        //EAST
    c++;
  else if(heading == 1)   //NORTH
    r--;
  else if(heading == 2)   //WEST
    c--;
  else if(heading == 3)   //SOUTH
    r++;
  else if(heading == 0.5) //NORTHEAST
  {
    c++;
    r--;
  }
  else if(heading == 1.5) //NORTHWEST
  {
    c--;
    r--;
  }
  else if(heading == 2.5) //SOUTHWEST
  {
    c--;
```

```
      r++;
    }
    else if(heading == 3.5) //SOUTHEAST
    {
      c++;
      r++;
    }
    else
      print("Error!!!"); //Soon to be changed.
    int[] retval = {
      c,r    };
    return retval;
  }
  void turn(double theta)
  {
    if(theta == left)
      heading++;
    else if(theta == right)
      heading--;
    else if(theta == halfLeft)
      heading += .5;
    else if(theta == halfRight)
      heading -= .5;
    if(heading >= 4)
      heading -= 4;
    if(heading < 0)
      heading += 4;
  }
}
```

2.1.2 Random Angle ALM

```
File: ALMSim/Robot.pde
class Robot
{
  private final double left      =  PI/2.0;
  private final double right     = -PI/2.0;
  private final double halfLeft  =  PI/4.0;
  private final double halfRight = -PI/4.0;
  private double drow, dcol;
  private int row, col;   //current location on grid
  private int speed;      //pixels per second
  private int overlap;    //a counter used when determining how far to overlap before tu
  private int roboSize;   //size in pixels
  private int viewRange;  //distance sensors can reach in pixels
  private double heading; //direction in polar grid
  private String type;    //circular or rectangular size
  private int[] roboColor = {0,0,220};

  //Constructors
  public Robot(int r, int c, int sp, double h, int sz, int vr, String t, int[] cl)
  {
    row       =  r;
    col       =  c;
    drow      =  r;
    dcol      =  c;
    speed     = sp;
    heading   =  h;
    roboSize  = sz;
    viewRange = vr;
    type      =  t;
    roboColor = cl;
  }
  public Robot(int r, int c, int sp, double h, int sz, int vr)
  {
    row       =  r;
    col       =  c;
    drow      =  r;
    dcol      =  c;
    speed     = sp;
    heading   =  h;
    roboSize  = sz;
    viewRange = vr;
    type  = "rect";
  }
  public Robot(int r, int c, int sp, double h)
  {
    row       =  r;
    col       =  c;
    drow      =  r;
    dcol      =  c;
    speed     = sp;
    heading   =  h;
    roboSize  =  5;
    viewRange =  3;
    type  = "rect";
  }
  public Robot()
  {
    row       = 350;
    col       = 5;
    drow      = row;
    dcol      = col;
    speed     = 100;
    heading   = PI/5.0;
    roboSize  = 5;
    viewRange = 3;
    type = "rect";
  }

  //Accessors
  int[] getPos()
  {
    int[] ret = {row, col};
    return ret;
  }
  int getSpeed()
  {
    return speed;
  }
  double getHeading()
  {
    return heading;
  }
  int getSize()
  {
    return roboSize;
  }
  int getRange()
  {
    return viewRange;
  }
  String getType()
  {
    return type;
  }
  int[] getColor()
  {
    return roboColor;
  }

  //"Setters"
  void setPos(int r, int c)
  {
    row = r;
    col = c;
  }
  void setSpeed(int sp)
```

```
{
  speed = sp;
}
void setHeading(int h)
{
  heading = h;
}
void setSize(int sz)
{
  roboSize = sz;
}
void setRange(int vr)
{
  viewRange = vr;
}
void setType(String t)
{
  type = t;
}
void setColor(int[] c)
{
  roboColor = c;
}

//Instance Methods
void Draw()
{
  /*for(int r=-roboSize; r<roboSize; r++)
    for(int c=-roboSize; c<roboSize; c++)
      if(row+r<0 || col+c<0 || row+r>=HEIGHT || col+c>=WIDTH)
        continue;
      else
        floorType[row+r][col+c] = 4;*/
  rectMode(CENTER);
  stroke(255,0,0);
  //stroke(roboColor[0],roboColor[1],roboColor[2]);
  fill(roboColor[0],roboColor[1],roboColor[2]);
  // find four corner points
  beginShape();
  for(double theta=PI/4.0; theta<=7*PI/4.0; theta+=PI/2.0)
    vertex((float)dcol+MARGIN+roboSize*sqrt(2.0)*cos((float)(heading+theta)),(float)dr
  endShape(CLOSE);
  //rect(col+MARGIN,row+MARGIN,roboSize*2,roboSize*2);
  stroke(255);
  fill(255);
  ellipse((float)dcol+MARGIN+roboSize*cos((float)heading),(float)drow+MARGIN+roboSize*
  //noFill();
}

void cutForward()
{
  //check next line of pixels only.
  //vector -> check 3 points
  boolean flag = false;
  for(int x=0; x<speed; x++)
  {
    // Each step I want to:
    // (1) Get the floor values in front of me
    // (2) Turn if they are obstacles
    // (3) Cut the grass and move one pixel if they are not.
    double[] objsInFront = checkForward(); // returns -1,0,1 relating to which pixel f
    if(objsInFront != null)
    {
      if(objsInFront[0] == 3 || objsInFront[1] == 3 || objsInFront[2] == 3)
      {
        turn(random(2.0*PI));
        break;
      }
      else
      {
        int R = roboSize;
        for(int r=-R; r<=R; r++)
          for(int c=-R; c<=R; c++)
            if(c*c+r*r<=R*R && row+r>0 && row+r<HEIGHT && col+c>0 && col+c<WIDTH)
              floorType[row+r][col+c] = 1;
        double[] pos = forward(dcol,drow);
        dcol = pos[0];
        drow = pos[1];
        col = (int)dcol;
        row = (int)drow;
        overlap = 0;
      }
    }
  }
  //draw robot again
  //this.Draw();
}
double[] checkForward()
{
  //shit time to change all this
  int[] x = new int[3];
  int[] y = new int[3];
  int R = roboSize;
  //front to the left
  x[0] = (int)(dcol+R*sqrt(2)*cos((float)(heading+PI/4.0)));
  y[0] = (int)(drow+R*sqrt(2)*-sin((float)(heading+PI/4.0)));
  //directly in front
  x[1] = (int)(dcol+R*cos((float)heading));
  y[1] = (int)(drow+R*-sin((float)heading));
  //front to the right
  x[2] = (int)(dcol+R*sqrt(2)*cos((float)(heading-PI/4.0)));
  y[2] = (int)(drow+R*sqrt(2)*-sin((float)(heading-PI/4.0)));

  //Check the above three points 1 pixel in front of the robot to see if it is clear o
  double[] retval = new double[3];
  for(int i=0; i<3; i++)
  {
    x[i] += 2*cos((float)heading);
    //println(2*cos((float)heading));
    y[i] -= 2*sin((float)heading);
    if(x[i]<0 || x[i]>=WIDTH || y[i]<0 || y[i]>=HEIGHT) // Treat boundaries as obstacl
      retval[i] = 3;//i;
    else
      retval[i] = floorType[y[i]][x[i]];
    /*if(floorType[y[i]][x[i]] == 3 || floorType[y[i]][x[i]] == 1) // If there is an o
      return i-1;*/
  }
  return retval;
}
double[] forward(double cola, double rowb)
{
  double c = cola;
  double r = rowb;
  c += cos((float)heading);
  r -= sin((float)heading);
  double[] retval = {c,r};
  return retval;
}
/*int[] forward(int cola, int rowb)
{
  int c = cola;
  int r = rowb;
  if(heading == 0)        //EAST
    c++;
  else if(heading == 1)   //NORTH
    r--;
  else if(heading == 2)   //WEST
    c--;
  else if(heading == 3)   //SOUTH
    r++;
  else if(heading == 0.5) //NORTHEAST
  {
    c++;
    r--;
  }
  else if(heading == 1.5) //NORTHWEST
  {
    c--;
    r--;
  }
  else if(heading == 2.5) //SOUTHWEST
  {
    c--;
    r++;
  }
  else if(heading == 3.5) //SOUTHEAST
  {
    c++;
    r++;
  }
  else
    print("Error!!!"); //Soon to be changed.
  int[] retval = {c,r};
  return retval;
}*/
/*void turn(double theta)
{
  if(theta == left)
    heading++;
  else if(theta == right)
    heading--;
  else if(theta == halfLeft)
    heading += .5;
  else if(theta == halfRight)
    heading -= .5;
  if(heading >= 4)
    heading -= 4;
  if(heading < 0)
    heading += 4;
}*/
void turn(double theta)
{
  heading+=theta;
}
```

```
}
```

2.1.3 Spiral ALM

```
class Robot
{
  private final double left      =  PI/2.0;
  private final double right     = -PI/2.0;
  private final double halfLeft  =  PI/4.0;
  private final double halfRight = -PI/4.0;
  private double gWidth, gHeight; //grid width and height, respectively
  private double drow, dcol; //current location in coordinate system
  private double nrow, ncol; //projected next location in coordinate system
  private double magnitude = ((double)WIDTH-10.0)/2.0;
  private float theta     = 0.0;
  private int cycle = 1;
  private int row, col;   //current location in matrix
  private int lrow, lcol; //last location in matrix
  private int speed;      //pixels per second
  private int overlap;    //a counter used when determining how far to overlap before tu
  private int roboSize;   //size in pixels
  private int viewRange;  //distance sensors can reach in pixels
  private double heading; //direction in polar grid
  private String type;    //circular or rectangular size
  private int[] roboColor = {0,0,220};

  //Constructors
  public Robot(int r, int c, int sp, double h, int sz, int vr, String t, int[] cl)
  {
    row       =  r;
    col       =  c;
    drow      =  r;
    dcol      =  c;
    speed     = sp;
    heading   =  h;
    roboSize  = sz;
    viewRange = vr;
    type      =  t;
    roboColor = cl;
  }
  public Robot(int r, int c, int sp, double h, int sz, int vr)
  {
    row       =  r;
    col       =  c;
    drow      =  r;
    dcol      =  c;
    speed     = sp;
    heading   =  h;
    roboSize  = sz;
    viewRange = vr;
    type  = "rect";
  }
  public Robot(int r, int c, int sp, double h)
  {
    row       =  r;
    col       =  c;
    drow      =  r;
    dcol      =  c;
    speed     = sp;
    heading   =  h;
    roboSize  =  5;
    viewRange =  3;
    type  = "rect";
  }
  public Robot()
  {
    row       = HEIGHT/2;
    col       = WIDTH/2;
    drow      = row;
    dcol      = col;
    speed     = 100;
    heading   = PI/2.0;
    roboSize  = 5;
    viewRange = 3;
    type = "rect";
  }

  //Accessors
  int[] getPos()
  {
    int[] ret = {row, col};
    return ret;
  }
  int getSpeed()
  {
    return speed;
  }
  double getHeading()
  {
    return heading;
  }
  int getSize()
```

```
{
  return roboSize;
}
int getRange()
{
  return viewRange;
}
String getType()
{
  return type;
}
int[] getColor()
{
  return roboColor;
}

//"Setters"
void setPos(int r, int c)
{
  row = r;
  col = c;
}
void setSpeed(int sp)
{
  speed = sp;
}
void setHeading(int h)
{
  heading = h;
}
void setSize(int sz)
{
  roboSize = sz;
}
void setRange(int vr)
{
  viewRange = vr;
}
void setType(String t)
{
  type = t;
}
void setColor(int[] c)
{
  roboColor = c;
}
```

```
//Instance Methods
void Draw()
{
  /*for(int r=-roboSize; r<roboSize; r++)
    for(int c=-roboSize; c<roboSize; c++)
      if(row+r<0 || col+c<0 || row+r>=HEIGHT || col+c>=WIDTH)
        continue;
      else
        floorType[row+r][col+c] = 4;*/
  rectMode(CENTER);
  stroke(255,0,0);
  //stroke(roboColor[0],roboColor[1],roboColor[2]);
  fill(roboColor[0],roboColor[1],roboColor[2]);
  // find four corner points
  beginShape();
  for(double thet=PI/4.0; thet<=7*PI/4.0; thet+=PI/2.0)
    vertex((float)dcol+MARGIN+roboSize*sqrt(2.0)*cos((float)(heading+thet)),(float)drow
  endShape(CLOSE);
  //rect(col+MARGIN,row+MARGIN,roboSize*2,roboSize*2);
  stroke(255);
  fill(255);
  ellipse((float)dcol+MARGIN+roboSize*cos((float)heading),(float)drow+MARGIN+roboSize*
  //noFill();
}

void cutForward()
{
  //check next line of pixels only.
  //vector -> check 3 points
  boolean flag = false;
  lrow = row;
  lcol = col;
  for(int x=0; x<speed; x++)
  {
    // Each step I want to:
    // (1) Get the floor values in front of me
    // (2) Turn if they are obstacles
    // (3) Cut the grass and move one pixel if they are not.
    // rcos(theta) - n*h/(2.0*PI)*theta
    //ncol = WIDTH/2.0+(magnitude-roboSize*2.0*(theta+2.0*PI/1000.0)/(2.0*PI))*cos(the
    //nrow = HEIGHT/2.0-(magnitude-roboSize*2.0*(theta+2.0*PI/1000.0)/(2.0*PI))*sin(th
    //println(""+ncol+" "+nrow);
    /*double[] objsInFront = checkForward();
    if(objsInFront != null)
    {
      if(objsInFront[0] == 3 || objsInFront[1] == 3 || objsInFront[2] == 3)
      {
        //turn(random(2.0*PI));
        break;
      }
      else
      {*/
        int R = roboSize;
        for(int r=-R; r<=R; r++)
          for(int c=-R; c<=R; c++)
            if(c*c+r*r<=R*R && row+r>0 && row+r<HEIGHT && col+c>0 && col+c<WIDTH)
              floorType[row+r][col+c] = 1;
        double originalRadius = 200;
        //turn(atan2((float)roboSize*theta/PI,originalRadius));
        //println(atan2((float)roboSize*theta/PI,originalRadius));
        //if(!checkRight())
        turn(1.0/originalRadius+theta*PI/1000.0);//originalRadius*theta*PI/1000.0);
        double[] pos = forward(dcol,drow);
        dcol = pos[0];
        drow = pos[1];
        //dcol = WIDTH/2.0+theta*cos(theta);
        //drow = HEIGHT/2.0+theta*sin(theta);
        col = (int)dcol;
        row = (int)drow;
        overlap = 0;
        theta+=PI/1000.0;
        cycle = (int)(theta/(2.0*PI))+1;
        //println(cycle);
      //}
    //}
  }
  //draw robot again
  //this.Draw();
}
double[] checkForward()
{
  //shit time to change all this
  int[] x = new int[3];
  int[] y = new int[3];
  int R = roboSize;
  //front to the left
  x[0] = (int)(dcol+R*sqrt(2)* cos((float)(heading+PI/4.0)));
  y[0] = (int)(drow+R*sqrt(2)*-sin((float)(heading+PI/4.0)));
  //directly in front
  x[1] = (int)(dcol+R* cos((float)heading));
  y[1] = (int)(drow+R*-sin((float)heading));
  //front to the right
  x[2] = (int)(dcol+R*sqrt(2)* cos((float)(heading-PI/4.0)));
  y[2] = (int)(drow+R*sqrt(2)*-sin((float)(heading-PI/4.0)));

  //Check the above three points 1 pixel in front of the robot to see if it is clear o
  double[] retval = new double[3];
  for(int i=0; i<3; i++)
  {
    x[i] += 2*cos((float)heading);
    //println(2*cos((float)heading));
    y[i] -= 2*sin((float)heading);
    if(x[i]<0 || x[i]>=WIDTH || y[i]<0 || y[i]>=HEIGHT) // Treat boundaries as obstacl
      retval[i] = 3;//i;
    else
      retval[i] = floorType[y[i]][x[i]];
    /*if(floorType[y[i]][x[i]] == 3 || floorType[y[i]][x[i]] == 1) // If there is an o
      return i-1;*/
  }
  return retval;
}
boolean checkRight() //grass to the right is cut/uncut
{
  int R = roboSize;
  int x = (int)(dcol+R*sqrt(2)* cos((float)(heading-PI/4.0))) + (int)cos((float)(headi
  int y = (int)(drow+R*sqrt(2)*-sin((float)(heading-PI/4.0))) - (int)sin((float)(headi
  if(floorType[y][x] == 1)
    return true;
  return false;
}
double[] forward(double cola, double rowb)
{
  double c = cola;
  double r = rowb;
  c += cos((float)heading);
  r -= sin((float)heading);
  double[] retval = {c,r};
  return retval;
}
void turn(double thet)
{
  heading+=thet;
}
}
```

2.2 Discussion

The Robot class is a private class included within the ALMSim class. The Robot contains various methods used to access and set its private variables. The class also has four methods, `Draw, turn, cutForward,` and `forward,` that aid in the moving process. Before moving forward, the Robot will check to see if it is at a boundary or an obstacle. If so, it turns left, otherwise, it will continue in the current heading. The Robot class allows for movement in 8 directions: N, NW, W, SW, S, SE, E, and NE. Whenever the Robot moves forward, it replaces the pixel values at where it was to "1", the value for cut grass. The Spiral ALM robot is *supposed* to move in a perfect spiral—one in which it covers everything with little to no overlap. This is currently not the case.