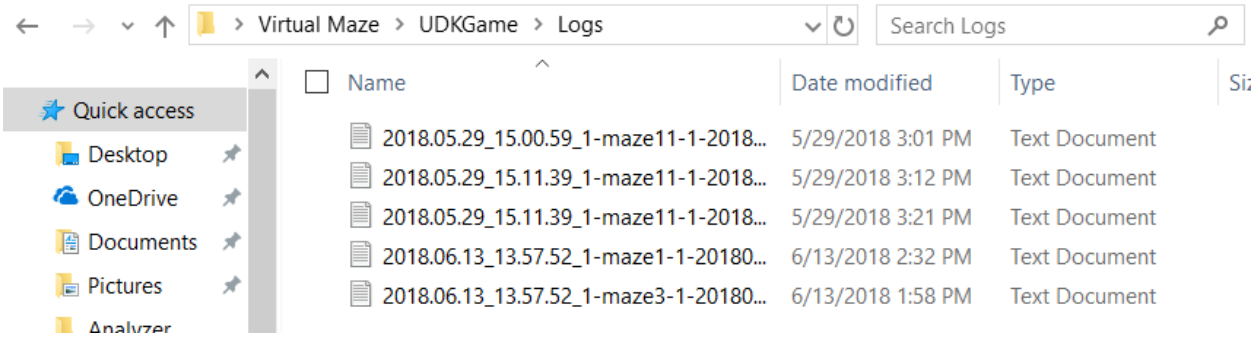
Sorawit Roongruengratanakul - roongrus@lafayette.edu
Professor Lisa Gabel & Professor Yih-Choung Yu
Summer 2018 EXCEL Project
Virtual Maze Analyzer Project Report

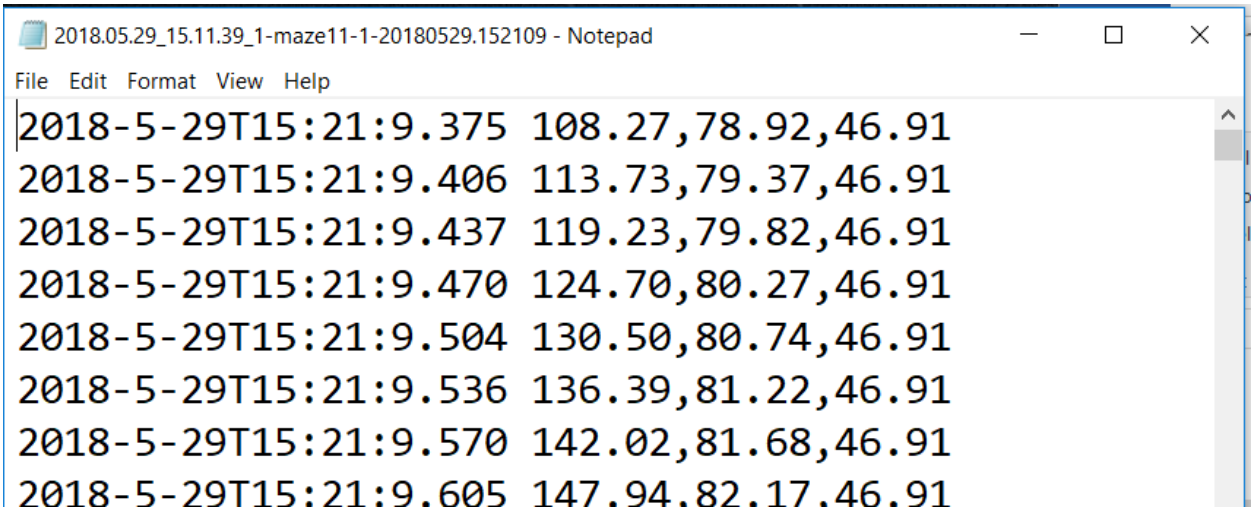
Using the Analyzer

To obtain log files, go to #GAME_ROOT_FOLDER# -> UDKGame -> Logs



Format of log file name:
Date_Time_[Group Number]-maze[Maze Number]-[Trial Number]-……
Group Number ranges from 1-2, Maze Number 1-12, and Trial Number 1-6.

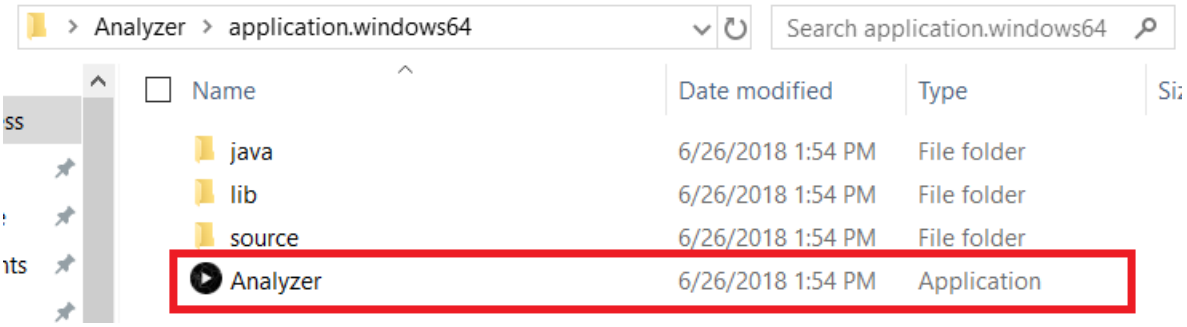
Inside of each log file:



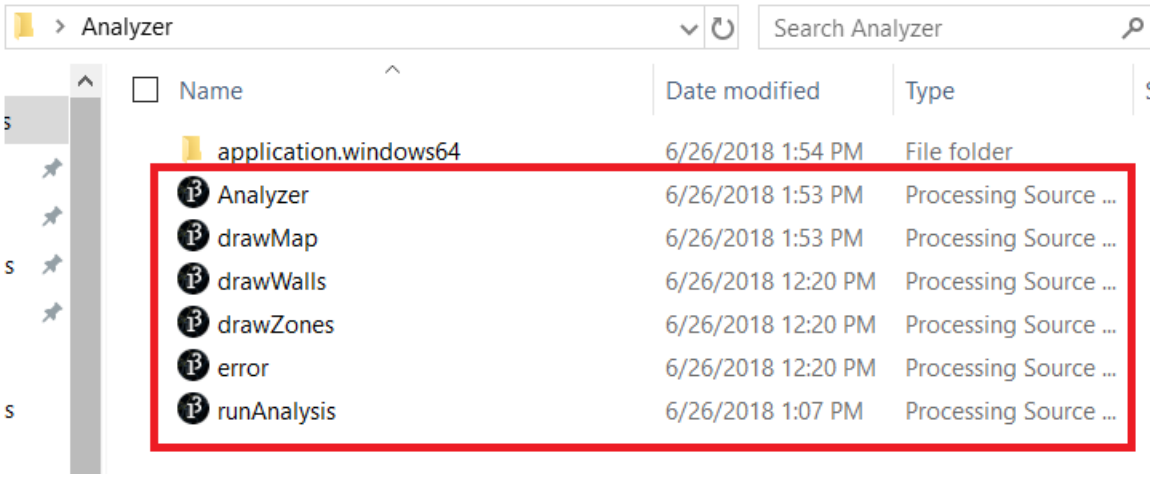
The format is as follows:
Year-Month-DateTHour:Minute:Second.Milliseconds X-Position, Y-Position, N/A

The coordinate system runs from (0,0) at the bottom right corner to (1536,1536) at the top left corner. The whole map can be divided into a 6x6 grid, each grid containing 256x256 units.

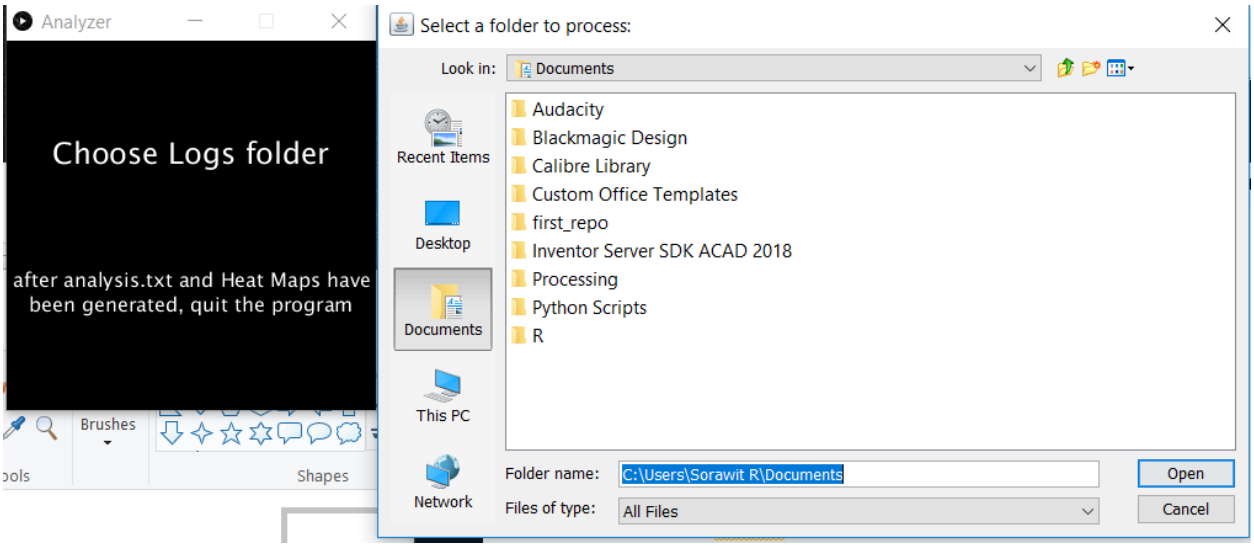
The Java Application (shown below) reads all the logs file in the "Logs" folder and outputs a summary text file called "analysis.txt" and a folder called "Heat Maps," which contains ONE heat map PER each log file.



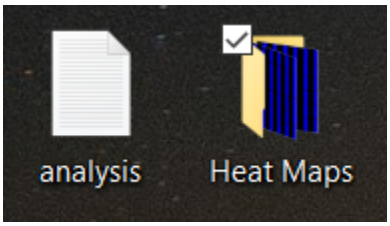
To edit the source code, edit one of the Java files below. Note: Processing needs to be installed. The program is divided into several files just for readability. It would still run if combined into one single file.



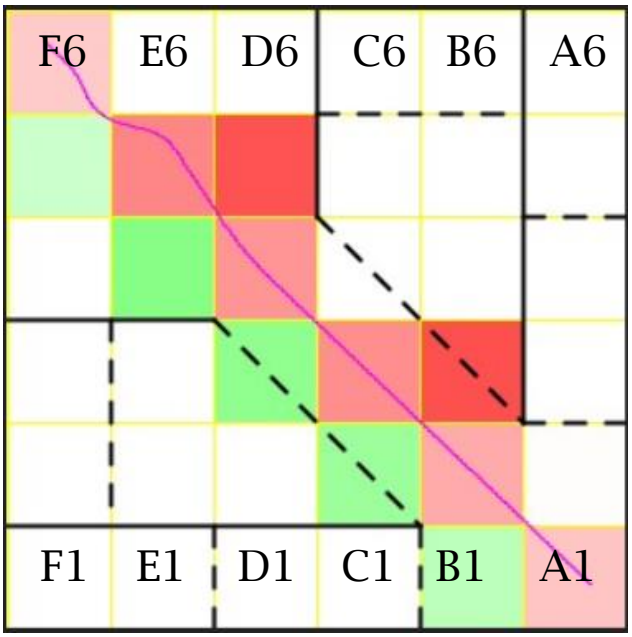
After running the program, you will see this pop-up folder selector. Select the “Logs” folder then hit “Open”. The program will take a moment to run.



After running, you will get this file and folder. Quit the program.



Inside “Heat Maps” folder:
This is a sample heat map. The purple line is the actual path taken, black solid line represents walls and dashed lines, danger zones. Each cell is colored differently according to its PE score, ranging form pure red(+3.0) to pure green(-3.0). White represents a PE score of 0.



ID	Group	Maze Num	Trial Numb	A1_Distan	B1_Distan	C1_Distan	D1_Distan	E1_Distan	F1_Distanc
2013.02.28_13.19.01	1	1	1	208.0942	38.15304	0	0	0	0
2013.02.28_13.19.01	1	1	2	165.0184	114.9866	0	0	0	0
2013.02.28_13.19.01	1	1	3	198.2821	43.04665	0	0	0	0
2013.02.28_13.19.01	1	1	4	213.143	0	0	0	0	0

1. Inside of analysis.txt, opened with Excel. The file details the distance travelled and time spent in each cell [A1 is bottom right and F6 is top left. A-F runs from right to left and 1-6 runs down to up.

TotalDista	TotalDurat	MeanSpee	FrozenTime
1922.414	10983	0.175035	0
1923.22	10959	0.175492	0
1913.795	10934	0.175032	0
1934.766	11056	0.174997	0
1929.293	11022	0.17504	0
1933.265	11018	0.175464	0
2617.468	22850	0.11455	15269

2. It also shows the total distance travelled in each maze, total duration(in milliseconds), mean speed(unit/msec) and FrozenTime, which is the total amount of time the participant spent standing still(also in msec).

3. Then it generates the average and standard deviation of distance/cell and time/cell PER MAZE. These numbers are used to convert the step 1 into Z-Scores.

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \qquad z = \frac{x - \mu}{\sigma}$$

$n =$ The number of data points

$\bar{x} =$ The mean of the x_i

$x_i =$ Each of the values of the data

Average				
Maze Number		1	198.3208	35.08957
Maze Number		2	0	0
Maze Number		3	0	0
Maze Number		4	0	0
Maze Number		5	160.1084	246.6701
Maze Number		6	159.956	4.578613
Maze Number		7	0	0
Maze Number		8	170.9933	22.84585
Maze Number		9	0	0
Maze Number		10	0	0
Maze Number		11	155.2081	252.9218
Maze Number		12	182.6808	61.60957

Z-Scores

ID	Group	Maze Num	Trial Numb	A1_Distan	B1_Distan
2013.02.28_13.19.01	1	1	1	0.449842	0.070847
2013.02.28_13.19.01	1	1	2	-1.53281	1.847719
2013.02.28_13.19.01	1	1	3	-0.00178	0.184017
2013.02.28_13.19.01	1	1	4	0.682223	-0.81149
2013.02.28_13.19.01	1	1	5	1.176376	-0.4796
2013.02.28_13.19.01	1	1	6	-0.77386	-0.81149

4. The PE (Performance Efficiency) score is calculated by averaging the Z-Score of distance and time.

5. The PE scores are then used to generate heat maps.

PE scores

ID	Group	Maze Num	Trial Numb	A1	B1	C1
2013.02.28_13.19.01	1	1	1	0.462171	0.075975	0
2013.02.28_13.19.01	1	1	2	-1.5256	1.848446	0
2013.02.28_13.19.01	1	1	3	0.03561	0.183251	0
2013.02.28_13.19.01	1	1	4	0.691512	-0.81127	0
2013.02.28_13.19.01	1	1	5	1.153939	-0.47703	0
2013.02.28_13.19.01	1	1	6	-0.80303	-0.81127	0
2013.02.28_13.19.01	1	11	1	0.792768	0.71308	-0.5193

6. Finally, the program also calculates the number of errors: number of times a participant enters a “danger zone” (dashed lines in heat map). Going in a zone then out counts as making ONE error.

Errors

ID	Group	Maze Num	Trial Number			
2013.02.28_13.19.01	1	1	1	Zone (1,3) - (1,5) : 0		Zone (2,5) - (2,6) : 0
2013.02.28_13.19.01	1	1	2	Zone (1,3) - (1,5) : 0		Zone (2,5) - (2,6) : 0
2013.02.28_13.19.01	1	1	3	Zone (1,3) - (1,5) : 0		Zone (2,5) - (2,6) : 0
2013.02.28_13.19.01	1	1	4	Zone (1,3) - (1,5) : 0		Zone (2,5) - (2,6) : 0
2013.02.28_13.19.01	1	1	5	Zone (1,3) - (1,5) : 0		Zone (2,5) - (2,6) : 0
2013.02.28_13.19.01	1	1	6	Zone (1,3) - (1,5) : 0		Zone (2,5) - (2,6) : 0

Code Full Explanation

```
/* Virtual Maze Analyzer
 Sorawit R.
 Summer 2018
 roongrus@lafayette.edu
 - generates an summary .txt file(open with Excel/spreadsheet programs)and heat maps for each individual .txt log file
 - needs Processing to run/edit this program
*/


/*
  cells in the virtual maze are labeled A1(bottom right)-F6(top left). A-F runs right to left and 1-6 runs bottom to top
  HOWEVER in the log files the maze is 1536*1536 (36 cells of 256*256 pixels each) where (0,0) is bottom right
  for example, (123,250) is in A1 and (1500,1200) is F5
*/

import java.util.Scanner;
import java.io.File;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.LineNumberReader;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;


int nfiles; //number of files in Logs folder

String id[]; //participant id
int group[]; //group number: 1=normal, 2=abnormal
int mazeNumber[]; //maze number (1-12)
String mazeTrial[]; //maze trial (1-6)


double distance[][]; //distance in each cell
long duration[][]; //duration in each cell
double avgDistance[][]; //avg distance per cell per maze
long avgDuration[][];  //avg duration per cell per maze
double stdDistance[][]; //standard deviation of distance per cell per maze
double stdDuration[][]; //standard deviation of duration per cell per maze
double zDistance[][]; //z-score of distance per cell per maze
double zDuration[][]; //z-score of duratino per cell per maze
double pe[][]; //performance efficiency score - average of zDistance and zDuration

double totalDistance[]; //total distance per log file
double totalDuration[]; //total duration per log file
```

import all necessary libraries

declare all global variables

```
double meanSpeed[]; //totalDistance divided by totalDuration
double frozenTime[]; //time participant stays still per log file

int mazeCount[]; //number of log files per maze

PrintWriter outFile; //output directory

String directory; //input directory

int size = 300; //size of heat map

PGraphics pg; //new graphics window
```

```
void folderSelected(File selection) //when a directory is selected
{
  if (selection == null)  //user doesn't select folder
    println("Window was closed or the user hit cancel.");
  else  //user selects folder
  {
    directory = selection.getAbsolutePath();
    println("User selected " + selection.getAbsolutePath());
    runAnalysis();  //get summary .txt file and .jpg heat maps in the same directory as log files
    drawMap(); //draws and saves heat map
    exit(); //automatically quits program
  }
}
```

```
void setup() {  //This function runs FIRST
  size(300,300); //300x300 window for instructions
  background(0);
  textAlign(CENTER);
  textSize(24);
  fill(255);
  text("Choose Logs folder",150,100);
  textSize(16);
  text("analysis.txt and Heat Maps folder",150,200);
  text("will be in the same directory as Logs",150,220);

  selectFolder("Select a folder to process:","folderSelected"); //select folder

  pg = createGraphics(300,300); //initiate PGraphics window for heat maps

}
```

```
//draws heat map
void drawMap()
{
  //directory in which heat maps to be printed
```

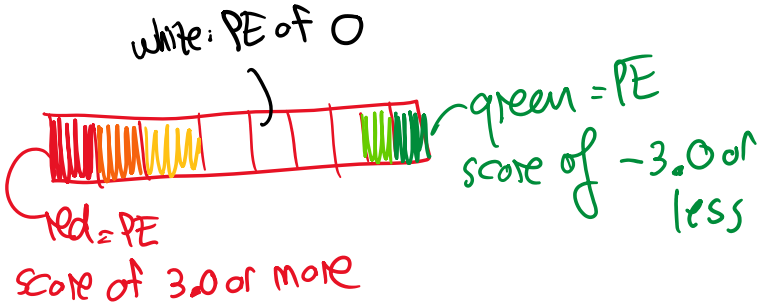
on-screen instructions

popup window to select folder directory

draw in background

```
File dir = new File(directory);
int c=0; //file index
try
{
 for(File file : dir.listFiles()) //for all files in directory
 {

  //go over all the log files again
  FileReader fr = new FileReader(file);
  LineNumberReader lnr = new LineNumberReader(fr);
  int nrows = 0;
  while(lnr.readLine() != null) } count number of rows in each file
   nrows++;
  lnr.close();

  //get x and y positions in each log file
  double xPos[] = new double[nrows];
  double yPos[] = new double[nrows];

  Scanner input = new Scanner(file);
  for(int i=0; i<nrows; i++)
  {
   String code = input.nextLine();
   xPos[i] = Double.parseDouble(code.substring(code.indexOf(' ')+1, code.indexOf(',')));
   yPos[i] = Double.parseDouble(code.substring(code.indexOf(',')+1, code.lastIndexOf(',')));
  }
  input.close();

  pg.beginDraw();
  pg.background(255);
  //yellow border
  pg.strokeWeight(1);
  pg.stroke(255,255,0); yellow

  for(int i=5; i>=0; i--)
  {
   for(int j=5; j>=0; j--)
   {
    double grad = pe[c][(5-j)+6*(5-i)]/3.0; how red/green the cell will be
    int col = (int)(grad*255);
    if(grad<=0)
     pg.fill(255+col,255,255+col); //draw green rectangles in case of negative PE score
    else
     pg.fill(255,255-col,255-col); //draw red rectangles in case of positive PE score
    pg.rect(j*size/6,i*size/6,size/6, size/6); //rectangle is 50x50 pixel each
   }
  }

  //purple stroke blue for visibility
  pg.stroke(0,0,255);
  pg.strokeWeight(2);
```

(319.25, 41.58)
yPos
xPos



global variable size is 300

```
      pg.noFill();
      pg.beginShape();
      //draws path
      for(int i=0; i<nrows; i++)
      {
        pg.vertex((float)(1536-xPos[i])*size/1536,(float)(1536-yPos[i])*size/1536);
      }
      pg.endShape();
      drawWalls(mazeNumber[c]); //draw walls using black lines
      drawZones(mazeNumber[c]); //draw danger zones using black dashed lines
      pg.endDraw();
      //save heat maps as .jpg files
      pg.save(directory.substring(0,directory.length()-4)+"Heat Maps/"+id[c]+"-"+group[c]+"-
"+mazeNumber[c]+"-"+mazeTrial[c]+".jpg");
      c++;
    }
  }
  //catch errors
  catch (FileNotFoundException e)
  {
    e.printStackTrace();
  }
  catch(IOException e)
  {
    e.printStackTrace();
  }
}

//draw walls using black lines
void drawWalls(int number) //number = maze number
{
  int unit = size/6;
  pg.strokeWeight(2);
  pg.stroke(0); //black stroke
  //walls are individual to each maze
  switch(number)
  {
    case 1:
      pg.line(0,3*unit,2*unit,3*unit); pg.line(0,5*unit,4*unit,5*unit); pg.line(3*unit,0,3*unit,2*unit);
pg.line(5*unit,0,5*unit,4*unit); break;
    case 2:
      pg.line(unit,0,unit,2*unit); pg.line(3*unit,0,3*unit,2*unit); pg.line(2*unit,3*unit,2*unit,6*unit);
pg.line(2*unit,3*unit,4*unit,3*unit); break;
    case 3:
      pg.line(unit,2*unit,6*unit,2*unit); pg.line(unit,2*unit,unit,4*unit);
pg.line(3*unit,3*unit,6*unit,3*unit); pg.line(3*unit,3*unit,3*unit,5*unit); break;
    case 4:
      pg.line(3*unit,unit,6*unit,unit); pg.line(3*unit,unit,3*unit,3*unit);
pg.line(3*unit,3*unit,4*unit,3*unit); pg.line(5*unit,3*unit,5*unit,4*unit);
pg.line(5*unit,4*unit,0,4*unit); break;
    case 5:
```

draw path

save as JPEGs

stack trace won't show when running program, only in Processing terminal

//walls depend on maze number

```
      pg.line(2*unit, 0, 2*unit, unit); pg.line(2*unit, unit, 4*unit, unit); pg.line(4*unit, unit, 4*unit,
5*unit); pg.line(0, unit, unit, unit); pg.line(unit, unit, unit, 3*unit); pg.line(unit, 3*unit, 3*unit,
3*unit); break;
    case 6:
      pg.line(0,unit,5*unit,unit); pg.line(0, 3*unit,4*unit, 3*unit); pg.line(4*unit, 3*unit, 4*unit,
5*unit); break;
    case 7:
      pg.line(0,unit,3*unit,unit); pg.line(4*unit,unit,6*unit,unit); pg.line(3*unit,unit,3*unit,3*unit);
pg.line(4*unit,unit,4*unit,4*unit); break;
    case 8:
      pg.line(unit,0,unit,3*unit); pg.line(unit,3*unit,3*unit,3*unit);
pg.line(4*unit,3*unit,5*unit,3*unit); pg.line(5*unit,3*unit,5*unit,4*unit);
pg.line(5*unit,4*unit,0,4*unit); break;
    case 9:
      pg.line(0,unit,5*unit,unit); pg.line(unit,unit,unit,4*unit); pg.line(unit,4*unit,3*unit,4*unit);
pg.line(2*unit,3*unit,6*unit,3*unit); pg.line(4*unit,3*unit,4*unit,5*unit);  break;
    case 10:
      pg.line(unit,0,unit,2*unit); pg.line(unit,2*unit,4*unit,2*unit);
pg.line(5*unit,2*unit,5*unit,4*unit); pg.line(5*unit,4*unit,0,4*unit); break;
    case 11:
      pg.line(0,unit,3*unit,unit); pg.line(3*unit,unit,3*unit,3*unit);
pg.line(2*unit,2*unit,2*unit,4*unit); pg.line(unit,3*unit,unit,5*unit);
pg.line(4*unit,unit,6*unit,unit); pg.line(4*unit,unit,4*unit,4*unit); break;
    case 12:
      pg.line(unit,0,unit,5*unit); pg.line(unit,5*unit,2*unit,5*unit);
pg.line(2*unit,unit,2*unit,4*unit); pg.line(4*unit,4*unit,4*unit,6*unit); break;
  }
}


//draw danger zones
void drawZones(int number) //number = maze number
{
  int unit = size/6; //there are 6x6 cells in each maze
  pg.noStroke();
  pg.fill(0);
  switch(number) //danger zones depend on maze number
   {
      case 1:
        dottedLine(1*unit,3*unit,1*unit,5*unit); dottedLine(2*unit,5*unit,2*unit,6*unit);
dottedLine(4*unit,5*unit,4*unit,6*unit); dottedLine(2*unit,3*unit,4*unit,5*unit);
dottedLine(3*unit,1*unit,5*unit,1*unit); dottedLine(5*unit,2*unit,6*unit,2*unit);
dottedLine(3*unit,2*unit,5*unit,4*unit); dottedLine(5*unit,4*unit,6*unit,4*unit); break;
      case 2:
        dottedLine(1*unit,1*unit,3*unit,1*unit); dottedLine(1*unit,2*unit,6*unit,2*unit);
dottedLine(0*unit,3*unit,2*unit,3*unit); dottedLine(3*unit,3*unit,3*unit,6*unit);
dottedLine(4*unit,3*unit,4*unit,6*unit); break;
      case 3:
        dottedLine(1*unit,0*unit,1*unit,2*unit); dottedLine(3*unit,2*unit,3*unit,3*unit);
dottedLine(1*unit,4*unit,3*unit,5*unit); dottedLine(3*unit,4*unit,6*unit,4*unit);
dottedLine(3*unit,5*unit,6*unit,5*unit); break;
```

```
      case 4:
        dottedLine(3*unit,0*unit,3*unit,1*unit); dottedLine(0*unit,2*unit,2*unit,4*unit);
dottedLine(3*unit,3*unit,5*unit,1*unit); dottedLine(3*unit,4*unit,3*unit,6*unit);
dottedLine(5*unit,4*unit,5*unit,6*unit); break;
      case 5:
        dottedLine(4*unit,0*unit,4*unit,1*unit); dottedLine(4*unit,3*unit,6*unit,3*unit);
dottedLine(4*unit,5*unit,6*unit,5*unit); dottedLine(0*unit,3*unit,1*unit,3*unit);
dottedLine(3*unit,3*unit,3*unit,6*unit); break;
      case 6:
        dottedLine(3*unit,1*unit,3*unit,6*unit); dottedLine(4*unit,1*unit,4*unit,3*unit);
dottedLine(4*unit,5*unit,4*unit,6*unit); break;
      case 7:
        dottedLine(0*unit,3*unit,2*unit,1*unit); dottedLine(4*unit,0*unit,4*unit,1*unit);
dottedLine(4*unit,3*unit,6*unit,3*unit); dottedLine(4*unit,4*unit,6*unit,4*unit);
dottedLine(0*unit,3*unit,3*unit,3*unit); dottedLine(3*unit,3*unit,3*unit,6*unit); break;
      case 8:
        dottedLine(1*unit,2*unit,3*unit,0*unit); dottedLine(3*unit,3*unit,6*unit,0*unit);
dottedLine(3*unit,4*unit,3*unit,6*unit); dottedLine(5*unit,4*unit,5*unit,6*unit); break;
      case 9:
        dottedLine(1*unit,3*unit,3*unit,1*unit); dottedLine(0*unit,4*unit,1*unit,4*unit);
dottedLine(3*unit,4*unit,3*unit,6*unit); dottedLine(4*unit,4*unit,6*unit,4*unit);
dottedLine(4*unit,5*unit,6*unit,5*unit); break;
      case 10:
        dottedLine(3*unit,0*unit,3*unit,2*unit); dottedLine(4*unit,0*unit,4*unit,2*unit);
dottedLine(3*unit,4*unit,3*unit,6*unit); dottedLine(5*unit,4*unit,5*unit,6*unit); break;
      case 11:
        dottedLine(2*unit,1*unit,2*unit,2*unit); dottedLine(4*unit,0*unit,4*unit,1*unit);
dottedLine(2*unit,4*unit,2*unit,6*unit); dottedLine(3*unit,3*unit,3*unit,6*unit);
dottedLine(4*unit,3*unit,6*unit,3*unit); dottedLine(4*unit,4*unit,6*unit,4*unit); break;
      case 12:
        dottedLine(2*unit,0*unit,2*unit,1*unit); dottedLine(2*unit,4*unit,2*unit,5*unit); break;
    }
}

//draws dotted lines. each line contains 15 dots
void dottedLine(int x1, int y1, int x2, int y2){
  double dist = Math.sqrt(Math.pow(x2-x1,2)+Math.pow(y2-y1,2));
  float steps = (float)dist/8;
 for(int i=0; i<=steps; i++) {
   float x = lerp(x1, x2, i/steps);
   float y = lerp(y1, y2, i/steps);
   pg.ellipse(x,y,3,3);
 }
}
```

// draw dotted lines; Processing doesn't have a function to draw dotted lines. each dot(3x3px) is 8 pixels apart

---

```
//counts errors in specified zone.
void error(int num, int x1, int y1, int x2, int y2) //num = log file number(from top), (x1,y1) and
(x2,y2) are endpoints of error zone. top left is (0,0) and bottom right is (6,6)
{
  File dir = new File(directory);
  int c=0;
```

//count number of errors

file index

```
try
{
  for (File file : dir.listFiles())
  {
    //go over all the log files again
    FileReader fr = new FileReader(file);
    LineNumberReader lnr = new LineNumberReader(fr);
    int nrows = 0;
    while (lnr.readLine() != null)
      nrows++;
    lnr.close();

    //get x and y coordinates
    if (c==num)
    {
      //get x and y positions in each log file
      double xPos[] = new double[nrows];
      double yPos[] = new double[nrows];

      Scanner input = new Scanner(file);
      for (int i=0; i<nrows; i++)
      {
        String code = input.nextLine();
        xPos[i] = Double.parseDouble(code.substring(code.indexOf(' ')+1, code.indexOf(',')));
        yPos[i] = Double.parseDouble(code.substring(code.indexOf(',')+1, code.lastIndexOf(',')));
      }
      input.close();

      int count =0; // count number of errors
      boolean goingIn; //check if going in or out. going in AND out counts as 1 error
      x1 = (6-x1)*256; //match coordinate system with log file's from (0,0) to (6,6) -> (1536,1536)
to (0,0)
      x2 = (6-x2)*256;
      y1 = (6-y1)*256;
      y2 = (6-y2)*256;
      if (x2-x1!=0) //slope is defined
      {
        double slope = (y2-y1)/(x2-x1);
        goingIn = true;
        for (int i=0; i<nrows-1; i++)
        {
          if ((xPos[i]>Math.min(x1, x2) && xPos[i]<Math.max(x1, x2))) //point-slope form
          {
            double y = slope*(xPos[i]-x1)+y1;
            if (y>Math.min(yPos[i], yPos[i+1]) && y<Math.max(yPos[i], yPos[i+1]))
            {
              if(goingIn==true)
              { count++; goingIn=false; }
              else
                goingIn=true;
            }
```

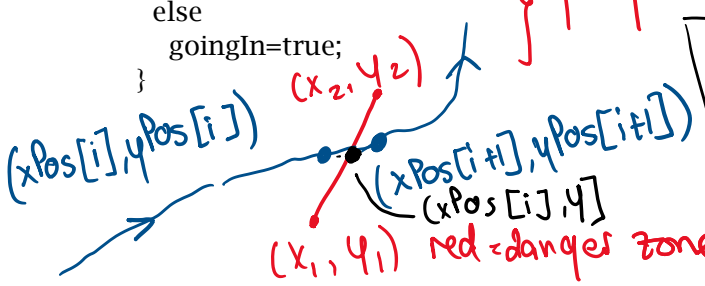
$y = mx + b$

point-slope form

$y - y_1 = m(x - x_1)$

$y = m(x - x_1) + y_1$

$m = \frac{y_2 - y_1}{x_2 - x_1}$



going in→out counts as 1 error

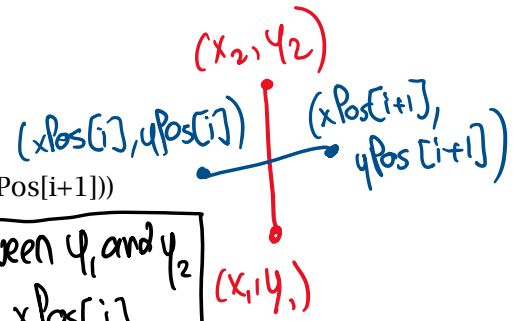
- scan all points
- if xPos[i] between X_1 and X_2 AND if y between yPos[i] and yPos[i+1]
  - means have crossed error zone

red = danger zone blue = actual path

```
        }
      }
    } else //slop is undefined (vertical error zone)
    {
      goingIn = true;
      for (int i=0; i<nrows-1; i++)
      {
        if (yPos[i]>Math.min(y1, y2) && yPos[i]<Math.max(y1, y2))
        {
          if (x1>Math.min(xPos[i], xPos[i+1]) && x1<Math.max(xPos[i], xPos[i+1]))
          {
            if(goingIn==true)
            { count++; goingIn=false; }
            else
              goingIn=true;
          }
        }
      }
    }
    //revert coordinates to original
    x1=-(x1/256-6);
    x2=-(x2/256-6);
    y1=-(y1/256-6);
    y2=-(y2/256-6);
    outFile.print("Zone ("+x1+","+y1+") - ("+x2+","+y2+") : "+count+"\t\t\t");
    break;
   }
   c++;
  }
 }
 //catch errors
 catch (FileNotFoundException e)
 {
  e.printStackTrace();
 }
 catch(IOException e)
 {
  e.printStackTrace();
 }
}
void runAnalysis(){ //generate analysis.txt
 nfiles = 0;
 try
 {
  //choose input folder
  File dir = new File(directory);

  //count number of files
  for(File file : dir.listFiles())
   nfiles++;
```

vertical error zone

in case slope = $\frac{y_2-y_1}{x_2-x_1} = \frac{\sim}{0}$ = undefined



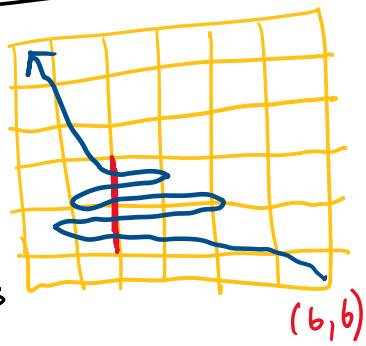
- if yPos[i] between y_1 and y_2

AND if x_1 between xPos[i] & xPos[i+1]

→ have crossed error zone

e.g. Zone (2,3) to (2,5) : 3

2 in-And-outs + 1 in-only



```
    //choose output file
    outFile = createWriter(directory.substring(0,directory.length()-4)+"analysis.txt");

    //initiate variables that store necessary info from each trial
    distance = new double[nfiles][36];
    duration = new long[nfiles][36];
    id = new String[nfiles];
    group = new int[nfiles];
    mazeNumber = new int[nfiles];
    mazeTrial = new String[nfiles];
    mazeCount = new int[12];
    totalDistance = new double[nfiles];
    totalDuration = new double[nfiles];
    meanSpeed = new double[nfiles];
    frozenTime = new double[nfiles];


    //print the headers print headers
    outFile.print("ID\tGroup\tMaze Number\tTrial
Number\tA1_Distance\tB1_Distance\tC1_Distance\tD1_Distance\tE1_Distance\tF1_Distance\tA
2_Distance\tB2_Distance\tC2_Distance\tD2_Distance\tE2_Distance\tF2_Distance\tA3_Distance
\tB3_Distance\tC3_Distance\tD3_Distance\tE3_Distance\tF3_Distance\tA4_Distance\tB4_Dista
nce\tC4_Distance\tD4_Distance\tE4_Distance\tF4_Distance\tA5_Distance\tB5_Distance\tC5_Di
stance\tD5_Distance\tE5_Distance\tF5_Distance\tA6_Distance\tB6_Distance\tC6_Distance\tD6
_Distance\tE6_Distance\tF6_Distance");

outFile.print("\tA1_Time\tB1_Time\tC1_Time\tD1_Time\tE1_Time\tF1_Time\tA2_Time\tB2_Ti
me\tC2_Time\tD2_Time\tE2_Time\tF2_Time\tA3_Time\tB3_Time\tC3_Time\tD3_Time\tE3_Ti
me\tF3_Time\tA4_Time\tB4_Time\tC4_Time\tD4_Time\tE4_Time\tF4_Time\tA5_Time\tB5_Ti
me\tC5_Time\tD5_Time\tE5_Time\tF5_Time\tA6_Time\tB6_Time\tC6_Time\tD6_Time\tE6_Ti
me\tF6_Time");
    outFile.print("\tTotalDistance\tTotalDuration\tMeanSpeed\tFrozenTime");
    outFile.println("");


    //for every log file in folder
    int count = 0;
    for(File file : dir.listFiles())
    {
      String name = file.getName();
      id[count] = name.substring(0,name.lastIndexOf('_'));
      group[count] = Integer.parseInt(name.substring(name.lastIndexOf('_')+1, name.indexOf('-')));
      mazeNumber[count] = Integer.parseInt(name.substring(name.indexOf('e')+1, name.indexOf('-
', name.indexOf('-')+1)));
      mazeCount[mazeNumber[count]-1]++;
      mazeTrial[count] = name.substring(name.indexOf('-', name.indexOf('-
')+1)+1,name.lastIndexOf('-'));
      //prints participant ID, group number, maze number, maze trial 1-2 1-12 1-6

outFile.print(id[count]+"\t"+group[count]+"\t"+mazeNumber[count]+"\t"+mazeTrial[count]+"\t");
```

```
    //count number of rows in each log file
    FileReader fr = new FileReader(file);
    LineNumberReader lnr = new LineNumberReader(fr);
    int nrows = 0;
    while(lnr.readLine() != null)
      nrows++;
    lnr.close();

    //get date, time, x and y positions from each log file
    String date[] = new String[nrows];
    String timeTemp[] = new String[nrows];
    double xPos[] = new double[nrows];
    double yPos[] = new double[nrows];
    Date time[] = new Date[nrows];
    DateFormat df = new SimpleDateFormat("hh:mm:ss.SSS");

    //within each log file, get time, x and y coordinates
    Scanner input = new Scanner(file);
    for(int i=0; i<nrows; i++)
    {
      String code = input.nextLine();
      date[i] = code.substring(0,code.indexOf('T'));
      timeTemp[i] = code.substring(code.indexOf('T')+1, code.indexOf(' '));
      time[i] = df.parse(timeTemp[i]);
      xPos[i] = Double.parseDouble(code.substring(code.indexOf(' ')+1, code.indexOf(',')));
      yPos[i] = Double.parseDouble(code.substring(code.indexOf(',')+1, code.lastIndexOf(',')));
    }
    input.close();


    //calculates distance and duration in each 'cell' per log file
    for(int i=1; i<nrows; i++)
    {
      int x = (int)xPos[i]/256;
      int y = (int)yPos[i]/256;
      distance[count][y*6+x]+=Math.sqrt(Math.pow(yPos[i]-yPos[i-1],2)+Math.pow(xPos[i]-xPos[i-
1],2));
      duration[count][y*6+x]+=time[i].getTime()-time[i-1].getTime();
      if((yPos[i]-yPos[i-1])+(xPos[i]-xPos[i-1])<5)
        frozenTime[count] += time[i].getTime()-time[i-1].getTime();
    }


    //prints distance and duration in each 'cell' per log file
    for(int i=0; i<36; i++)
    {
       totalDistance[count] += distance[count][i];
       outFile.print(distance[count][i]+"\t");
    }

    for(int i=0; i<36; i++)
```

frozen time tolerance: 5px

// prints A1 - F6 distance
A1 - F6 duration

```
    {
        totalDuration[count] += duration[count][i];
        outFile.print(duration[count][i]+"\t");
    }
    meanSpeed[count] = totalDistance[count]/totalDuration[count];

outFile.print(totalDistance[count]+"\t"+totalDuration[count]+"\t"+meanSpeed[count]+"\t"+froze
nTime[count]);
    outFile.println("");


    count++;
  }


  //calculates and prints average per maze
  avgDistance = new double[12][36];
  avgDuration = new long[12][36];
  for(int i=0; i<nfiles; i++)
  {
   for(int j=0; j<36; j++)
   {
    avgDistance[mazeNumber[i]-1][j] += distance[i][j];
    avgDuration[mazeNumber[i]-1][j] += duration[i][j];
   }
  }
  for(int i=0; i<12; i++)
  {
   for(int j=0; j<36; j++)
   {
    if(mazeCount[i]!=0) //avoid division by 0
    {
     avgDistance[i][j] /= mazeCount[i];
     avgDuration[i][j] /= mazeCount[i];
    }
   }
  }
  outFile.println("");
  outFile.print("Average");
  outFile.println("");
  for(int i=0; i<12; i++)
  {
   outFile.print("Maze Number\t\t"+(i+1)+"\t\t");
   for(int j=0; j<36; j++)
   {
    outFile.print(avgDistance[i][j]+"\t");
   }
   for(int j=0; j<36; j++)
   {
    outFile.print(avgDuration[i][j]+"\t");
   }
```

// prints total distance (ΣA1-F6 distance)
total duration (ΣA1-F6 duration)
mean speed (distance ÷ time)
frozen time (move less than 5 pixels)

GO OVER ALL FILES

distance = Σdistance / #files

duration = Σduration / #files

each maze has its own avg. distance and duration

// prints out all avg. distances & durations

```
    outFile.println("");
  }


  //calculates and prints standard deviation per maze
  stdDistance = new double[12][36];
  stdDuration = new double[12][36];
  for(int i=0; i<nfiles; i++)
  {
    for(int j=0; j<36; j++)
    {
      stdDistance[mazeNumber[i]-1][j] += Math.pow(distance[i][j]-avgDistance[mazeNumber[i]-
1][j],2.0);
      stdDuration[mazeNumber[i]-1][j] += Math.pow(duration[i][j]-avgDuration[mazeNumber[i]-
1][j],2.0);
    }
  }
  outFile.println("");
  outFile.print("SD");
  outFile.println("");
  for(int i=0; i<12; i++)
  {
    outFile.print("Maze Number\t\t"+(i+1)+"\t\t");
    for(int j=0; j<36; j++)
    {
      stdDistance[i][j] = Math.pow(stdDistance[i][j]/(mazeCount[i]-1),0.5);
      outFile.print(stdDistance[i][j]+"\t");
    }
    for(int j=0; j<36; j++)
    {
      stdDuration[i][j] = Math.pow(stdDuration[i][j]/(mazeCount[i]-1),0.5);
      outFile.print(stdDuration[i][j]+"\t");
    }
    outFile.println("");
  }


//calculates z scores (distance and duration separately) per maze
  zDistance = new double[nfiles][36];
  zDuration = new double[nfiles][36];
  for(int i=0; i<nfiles; i++)
  {
    for(int j=0; j<36; j++)
    {
      if(stdDistance[mazeNumber[i]-1][j]!=0)
      {
        zDistance[i][j] = (distance[i][j]-avgDistance[mazeNumber[i]-
1][j])/stdDistance[mazeNumber[i]-1][j];
        zDuration[i][j] = (duration[i][j]-avgDuration[mazeNumber[i]-
1][j])/stdDuration[mazeNumber[i]-1][j];
```

calculate standard deviations of distance/duration PER CELL PER MAZE.
(means 12 x 36 SD distances
12 x 36 SD durations
← 12 mazes, each with 6x6 cells

Sample SD = $\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$

converts original distances & durations into Z-scores

// Z-score = $\frac{x-\mu}{\sigma}$ ← avg, ← SD

```
      }
    }
  }
  //prints out z-scores
  outFile.println("");
  outFile.println("");
  outFile.println("Z-Scores");
  outFile.println("");
  outFile.println("");

  outFile.print("ID\tGroup\tMaze Number\tTrial
Number\tA1_Distance\tB1_Distance\tC1_Distance\tD1_Distance\tE1_Distance\tF1_Distance\tA
2_Distance\tB2_Distance\tC2_Distance\tD2_Distance\tE2_Distance\tF2_Distance\tA3_Distance
\tB3_Distance\tC3_Distance\tD3_Distance\tE3_Distance\tF3_Distance\tA4_Distance\tB4_Dista
nce\tC4_Distance\tD4_Distance\tE4_Distance\tF4_Distance\tA5_Distance\tB5_Distance\tC5_Di
stance\tD5_Distance\tE5_Distance\tF5_Distance\tA6_Distance\tB6_Distance\tC6_Distance\tD6
_Distance\tE6_Distance\tF6_Distance");

outFile.print("\tA1_Time\tB1_Time\tC1_Time\tD1_Time\tE1_Time\tF1_Time\tA2_Time\tB2_Ti
me\tC2_Time\tD2_Time\tE2_Time\tF2_Time\tA3_Time\tB3_Time\tC3_Time\tD3_Time\tE3_Ti
me\tF3_Time\tA4_Time\tB4_Time\tC4_Time\tD4_Time\tE4_Time\tF4_Time\tA5_Time\tB5_Ti
me\tC5_Time\tD5_Time\tE5_Time\tF5_Time\tA6_Time\tB6_Time\tC6_Time\tD6_Time\tE6_Ti
me\tF6_Time");
  outFile.println("");
  for(int i=0; i<nfiles; i++)
  {
    outFile.print(id[i]+"\t"+group[i]+"\t"+mazeNumber[i]+"\t"+mazeTrial[i]+"\t");
    for(int j=0; j<36; j++)
      outFile.print(zDistance[i][j]+"\t");

    for(int j=0; j<36; j++)
      outFile.print(zDuration[i][j]+"\t");
    outFile.println("");
  }


  //calculates performance efficiency scores per log file
  pe = new double[nfiles][36];
  for(int i=0; i<nfiles; i++)
  {
    for(int j=0; j<36; j++)
      pe[i][j] = (zDistance[i][j]+zDuration[i][j])/2;
  }

  //prints out performance efficiency scores
  outFile.println("");
  outFile.println("");
  outFile.println("PE scores");
  outFile.println("");
  outFile.println("");
```

prints out all z-scores

$$\text{PE score} = \frac{z_{\text{distance}} + z_{\text{duration}}}{2}$$

can be either + or −.

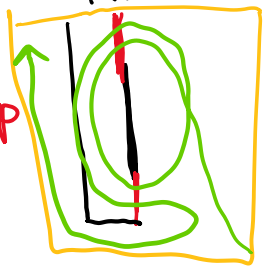
```
    outFile.print("ID\tGroup\tMaze Number\tTrial Number\tA1 \tB1 \tC1 \tD1 \tE1 \tF1 \tA2
\tB2 \tC2 \tD2 \tE2 \tF2 \tA3 \tB3 \tC3 \tD3 \tE3 \tF3 \tA4 \tB4 \tC4 \tD4 \tE4 \tF4 \tA5 \tB5
\tC5 \tD5 \tE5 \tF5 \tA6 \tB6 \tC6 \tD6 \tE6 \tF6 ");    outFile.println("");
    for(int i=0; i<nfiles; i++)
    {
      outFile.print(id[i]+"\t"+group[i]+"\t"+mazeNumber[i]+"\t"+mazeTrial[i]+"\t");
      for(int j=0; j<36; j++)
        outFile.print(pe[i][j]+"\t");
      outFile.println("");
    }

    //prints number of errors. top left corner is (0,0) and bottom right is (6,6)
    outFile.println("");
    outFile.println("");
    outFile.println("Errors");
    outFile.println("");
    outFile.println("");
    outFile.print("ID\tGroup\tMaze Number\tTrial Number");
    outFile.println("");
    for(int i=0; i<nfiles; i++)
    {
      outFile.print(id[i]+"\t"+group[i]+"\t"+mazeNumber[i]+"\t"+mazeTrial[i]+"\t");
      switch(mazeNumber[i])
      {
       case 1:
         error(i,1,3,1,5); error(i,2,5,2,6); error(i,4,5,4,6); error(i,2,3,4,5); error(i,3,1,5,1);
error(i,5,2,6,2); error(i,3,2,5,4); error(i,5,4,6,4); break;
       case 2:
         error(i,1,1,3,1); error(i,1,2,6,2); error(i,0,3,2,3); error(i,3,3,3,6); error(i,4,3,4,6); break;
       case 3:
         error(i,1,0,1,2); error(i,3,2,3,3); error(i,1,4,3,5); error(i,3,4,6,4); error(i,3,5,6,5); break;
       case 4:
         error(i,3,0,3,1); error(i,0,2,2,4); error(i,3,3,5,1); error(i,3,4,3,6); error(i,5,4,5,6); break;
       case 5:
         error(i,4,0,4,1); error(i,4,3,6,3); error(i,4,5,6,5); error(i,0,3,1,3); error(i,3,3,3,6); break;
       case 6:
         error(i,3,1,3,6); error(i,4,1,4,3); error(i,4,5,4,6); break;
       case 7:
         error(i,0,3,2,1); error(i,4,0,4,1); error(i,4,3,6,3); error(i,4,4,6,4); error(i,0,3,3,3);
error(i,3,3,3,6); break;
       case 8:
         error(i,1,2,3,0); error(i,3,3,6,0); error(i,3,4,3,6); error(i,5,4,5,6); break;
       case 9:
         error(i,1,3,3,1); error(i,0,4,1,4); error(i,3,4,3,6); error(i,4,4,6,4); error(i,4,5,6,5); break;
       case 10:
         error(i,3,0,3,2); error(i,4,0,4,2); error(i,3,4,3,6); error(i,5,4,5,6); break;
       case 11:
         error(i,2,1,2,2); error(i,4,0,4,1); error(i,2,4,2,6); error(i,3,3,3,6); error(i,4,3,6,3);
error(i,4,4,6,4); break;
       case 12:
         error(i,2,0,2,5); break;
```

prints out PE scores. They will later be used to color heat maps

//check for errors. see pgs 12-13 for "error" function

maze 12 special case
//counts 2 error zones as 1 since they form a loop

2 loops
2 errors
↓ MAZE 1



```
      }

      outFile.println("");
     }
     outFile.close();

    }
    //catch errors
    catch (FileNotFoundException e)
    {
      e.printStackTrace();
    }
    catch(IOException e)
    {
      e.printStackTrace();
    }
    catch(java.text.ParseException e)
    {
      println(e);
    }
  }
```