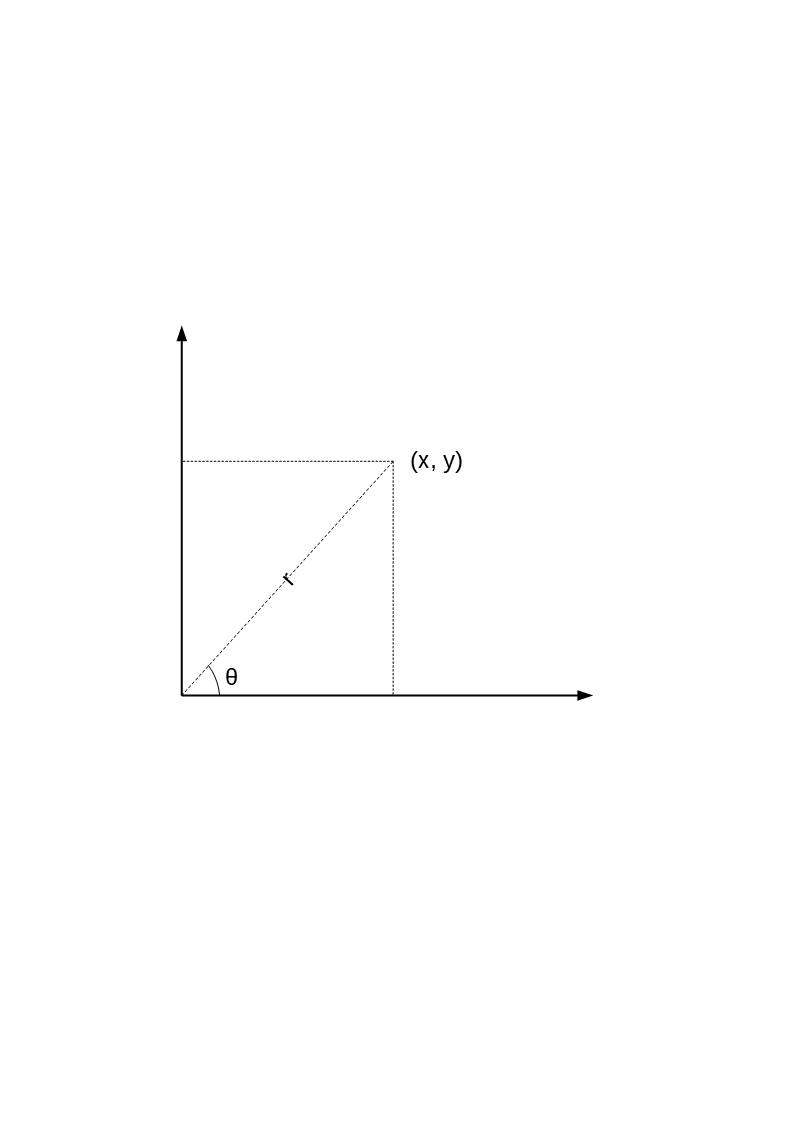
# Motivation

This is a simple yet exciting project idea I came up with in office. Recently, my colleague’s son fell ill with the dengue fever, and on kids the effect is huge. Their immunity is low, and platelets decrease rapidly below danger levels. Dengue is a mosquito spread disease, and mosquitoes are a big issue in India, and its spread and control is quite difficult due to insufficient data on the spread of mosquitoes.

Here, I present an idea that I am working on from the past weeks that could simulate the spread of dengue fever using a mosquito simulation program. So, let’s understand what we have and what we know and build upon it. In the simulation one major aspect is to code the random walk of the mosquitoes on a plane.

I took an approach of performing 2-dimensional random walks on a hypothetical plane using polar coordinates. Those of you who are unaware of polar coordinates, here is a small brief. Instead of using the generic (x, y) notation to locate a point on a plane we use an angle and distance (r,θ) approach.

This makes the simulation a problem. In this article we will discuss on this particular problem and look at some solutions I had come up with. 2-dimensional random walks are already done using an x-y grid in which the particle which is subjected to randomness can move in any 4,8,12 or 24 directions. Mostly people pick 4 directions and keep the step size so small that it seems the particle is moving in all directions including diagonals perfectly. This is a popular digital display trick heavily used today. Display devices are made with higher resolution to show diagonals and curves as accurately as possible but this accuracy is only perceived accuracy and not true accuracy. This is also complemented by anti-aliasing. This is a topic on digital display technologies, which means I’m going off topic.

Mosquitoes or any living insect tend to move away from their source or point of birth, they may visit it back but in-order to spread they must move away. Their spread is equal in all directions. Which means that the random walk must be biased. It can’t be constantly biased as they may find a chance to come back home after a farther travel. So, the distance the mosquito has traveled from home, or lets take displacement of the mosquito from home, which is easier to visualize, affects the biasing of the random walk. So, the closer the mosquito is to home the more biased the random walk would be to push it away from home, and the farther it is from home the lesser biased it would be.

Thus we have 2 problems:

1. 2-dimensional dynamically biased random walks
2. Using polar coordinates

This is only the tip of the iceberg, maybe only a part of it. And in the upcoming articles I will expand the picture and explain the possibilities of this simulation.

# 2-dimensional dynamically biased random walks

Here we use processing library as it is a simple and easy-to-use library written in Java. I am using this library only in this step.

We create 1000 mosquitoes first

int init\_n\_mosquittos = 1000;

ArrayList<Mosquitto> mosquittos = new ArrayList<Mosquitto>();

ArrayList<Mosquitto> next\_mosquittos = new ArrayList<Mosquitto>();

int new\_index = init\_n\_mosquittos;

int show\_after = 1;

int next\_show = 0;

void setup() {

size(960, 720);

frameRate(45);

for (int i = 0 ; i < init\_n\_mosquittos ; i++) {

mosquittos.add(new Mosquitto(i));

}

}

void draw() {

if(next\_show % show\_after == 0) {

background(220);

for (int i = 0 ; i < mosquittos.size() ; i++) {

mosquittos.get(i).show();

}

next\_show = 0;

}

next\_show++;

for (int i = 0 ; i < mosquittos.size() ; i++) {

mosquittos.get(i).update();

}

for (int i = 0 ; i < mosquittos.size() ; i++) {

if(mosquittos.get(i).kill) {

mosquittos.remove(i);

i--;

}

}

mosquittos.addAll(mosquittos.size()-1, next\_mosquittos);

if(next\_mosquittos.size() > 0) {

System.out.println("Next Mosquittos: " + next\_mosquittos.size());

System.out.println(" Mosquittos: " + mosquittos.size());

}

next\_mosquittos.clear();

}