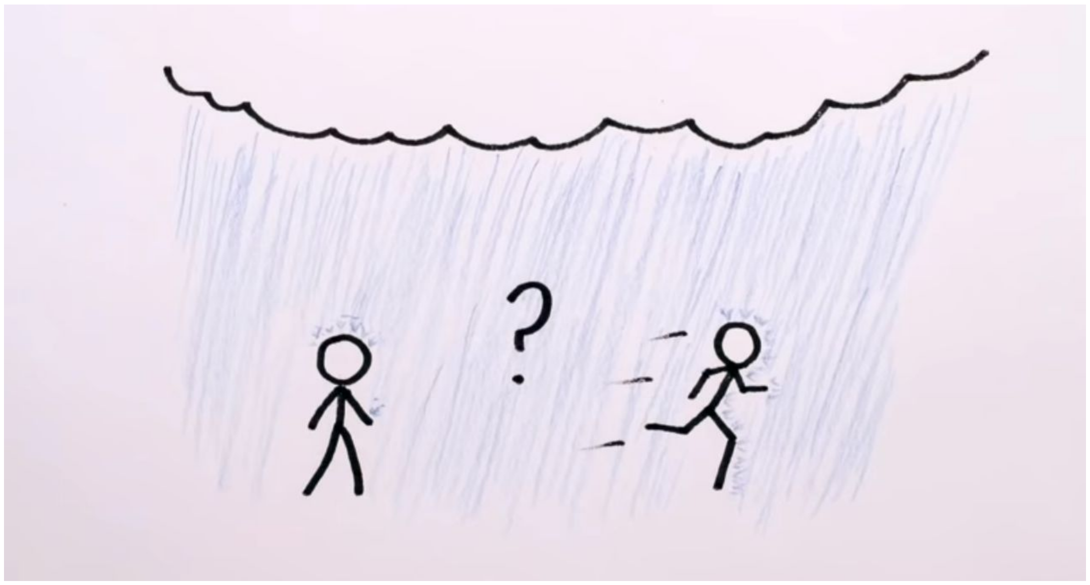
ISC 3313 Final Project
Abelardo Riojas FSUID: ar18aa

"Is it Better to Walk or Run in The Rain?"

Solved using Computational Simulation in C++.



Introduction

Living in Florida has its problems. There's the possibility of being eaten alive by an alligator, the blistering heat that serves as a monument of arrogance of mankind, and more mentally insane people in its population than any other state means the headline "Florida man eats another man alive" isn't really news around here. But perhaps the paramount issue that Florida citizens face is the state's extremely volatile weather patterns. Whether or not it is going to rain in Florida is a question no one can really solve with rhyme or reason, but only with pure, unadulterated luck. This leads to the very pragmatic question all Floridians have wanted an answer to for a long time: "Is it better to walk or run in the rain?"

From the onset of this problem, running seems to be the best course of action. Since it minimizes the amount of rain hitting you from the top due to being in the rain for less time. However, one might also think that walking could possibly be better than running when one considers that the runner will be hitting more raindrops from the *front* than he would be walking. This leads to the ambiguity of the problem, and the possibility of walking being better than running at certain conditions.

Background Research

In order to solve this problem, background research needed to be put into the values we're going to use in our simulation. Many of these values came from Anthropometrics, the study of the measurements of the human body.

Anthropometric Measurement	Measurement / m
Average Human Height (Male)	1.77
Average Human Side Length (Male)	.26
Average Human Broadness (Male)	.40

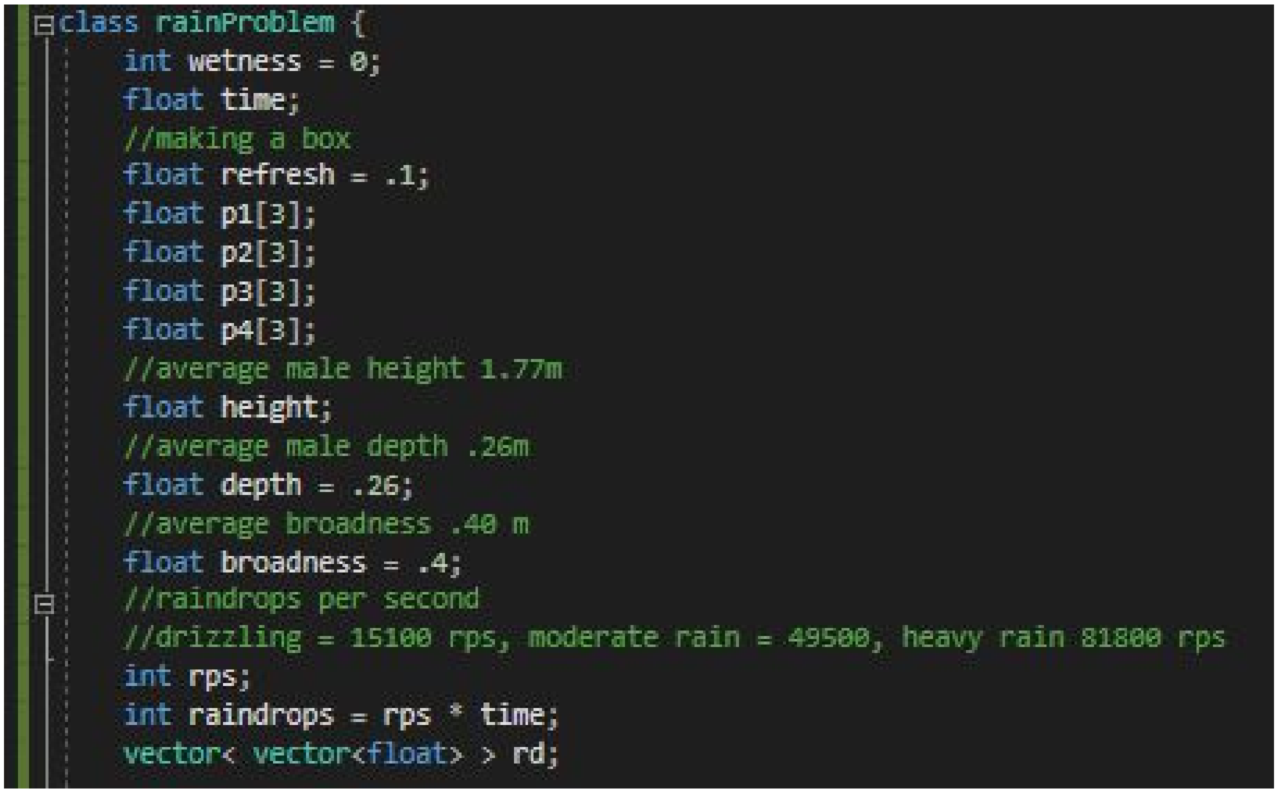
Source: Center for Disease Control and Prevention https://www.cdc.gov/nchs/fastats/body-measurements.htm

Other values needed in this simulation were: the amount of raindrops that hit the ground per square meter per second, the walking and running speed of humans, and the terminal velocity of raindrops.

Measurement	Value
Light rain[1]	151 raindrops s^{-1} m^{-2}
Moderate rain[1]	495 raindrops s^{-1} m^{-2}
Heavy Rain [1]	818 raindrops s^{-1} m^{-2}
Human walking speed [2]	1.4 m s^{-1}
Human running speed [2]	6.7 m s^{-1}
Terminal velocity of raindrops [3]	10 m s^{-1}

Sources: 1: http://www.kgbanswers.com/how-many-raindrops-fall-in-one-second/4191107 2: https://www.quora.com/What-is-the-average-running-speed-of-a-human 3: Terminal Velocity of Raindrops Aloft, G. B. Foote and P. S. Du Toit Institute of Atmospheric Physics, The University of Arizona, Tucson

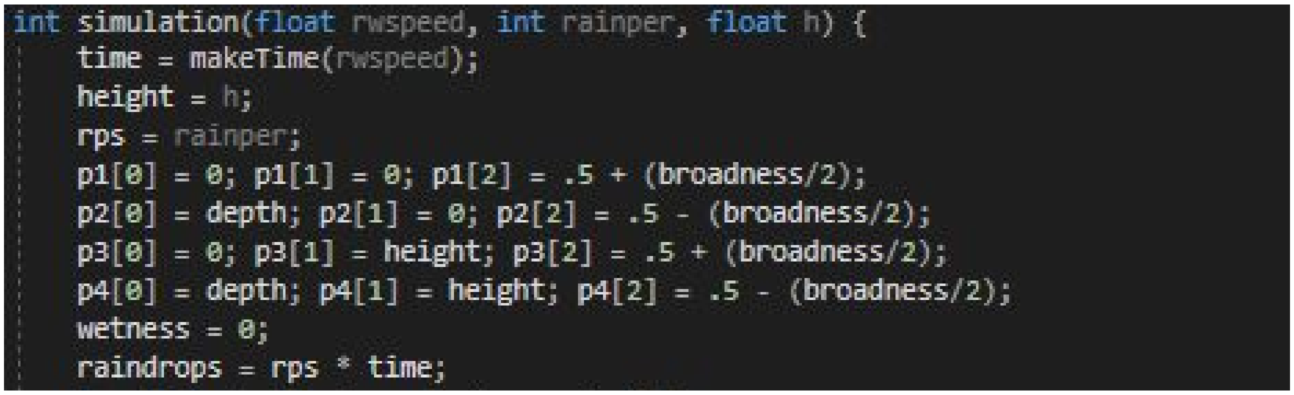
Our Key Players: The Private Variables



```
class rainProblem {
    int wetness = 0;
    float time;
    //making a box
    float refresh = .1;
    float p1[3];
    float p2[3];
    float p3[3];
    float p4[3];
    //average male height 1.77m
    float height;
    //average male depth .26m
    float depth = .26;
    //average broadness .40 m
    float broadness = .4;
    //raindrops per second
    //drizzling = 15100 rps, moderate rain = 49500, heavy rain 81800 rps
    int rps;
    int raindrops = rps * time;
    vector< vector<float> > rd;
```

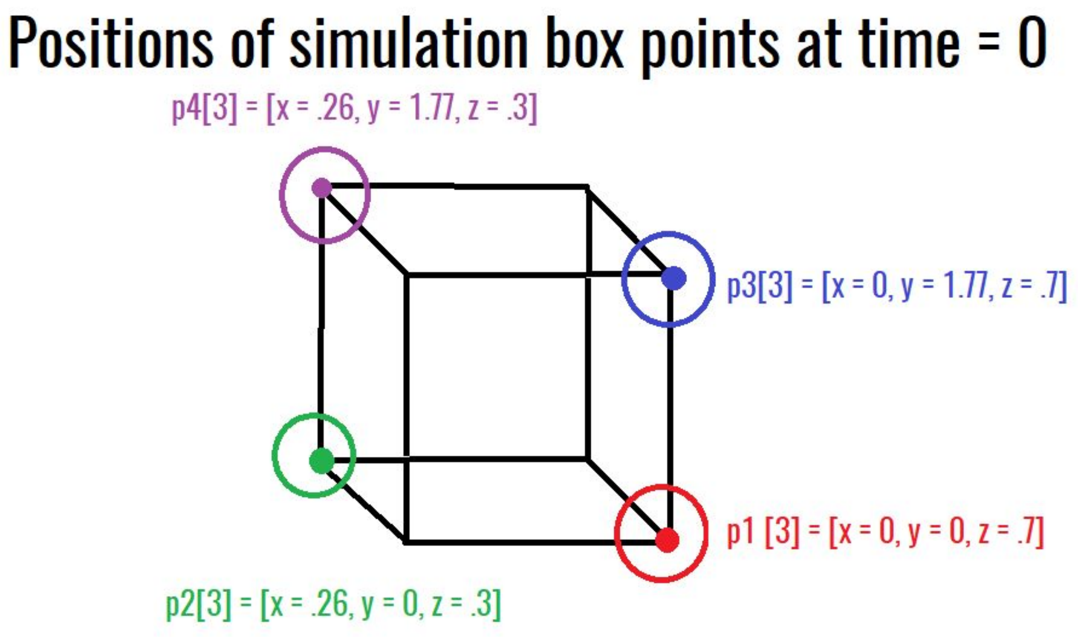
This is a screenshot containing the private variables of the rainProblem class that will be used to solve our problem. wetness is the integer we're going to return to compare how wet our runner gets, time is a float we use to correlate distance with iteration counts. refresh is a float we use to set the refresh rate of our simulation. The arrays p#[3] are used to store the x, y, and z positions of our human-sized box, height, depth, and broadness are used in calculating the position of our box, rps is an integer that contains the raindrops falling from the sky per second, vector<vector<float>> is a vector that contains the x, y, z, and trigger times for our raindrops in our simulation. All of these values are used to create our simulation.

int simulation() Part I

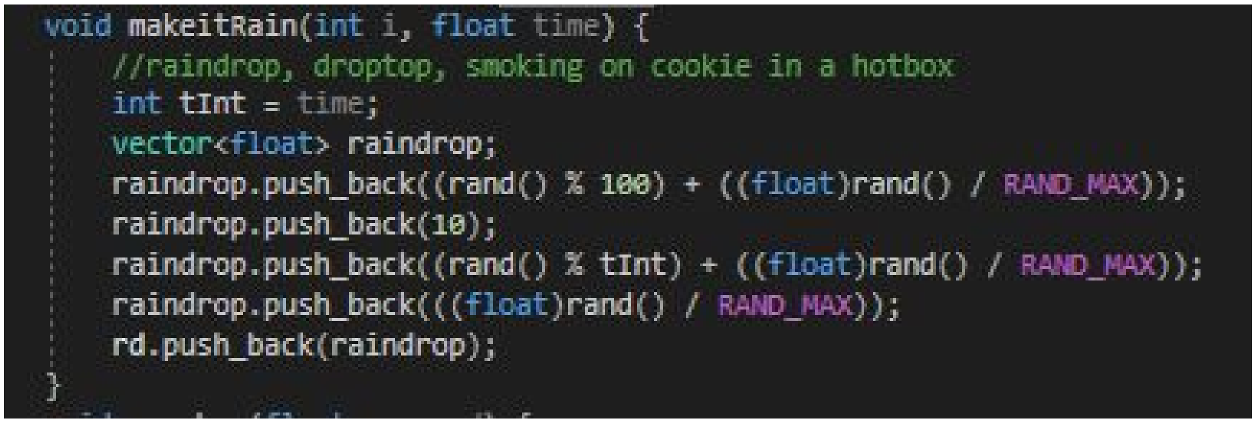


```
int simulation(float rwspeed, int rainper, float h) {
    time = makeTime(rwspeed);
    height = h;
    rps = rainper;
    p1[0] = 0; p1[1] = 0; p1[2] = .5 + (broadness/2);
    p2[0] = depth; p2[1] = 0; p2[2] = .5 - (broadness/2);
    p3[0] = 0; p3[1] = height; p3[2] = .5 + (broadness/2);
    p4[0] = depth; p4[1] = height; p4[2] = .5 - (broadness/2);
    wetness = 0;
    raindrops = rps * time;
```

Here we set the values of our private variables to what they need to be at the start of our simulation, as well as feed arguments to our private variables like speed (running or walking, raindrops per second, and height). In order to better understand the values of box position variables, see the below illustration.


Positions of simulation box points at time = 0
p4[3] = [x = .26, y = 1.77, z = .3]
p3[3] = [x = 0, y = 1.77, z = .7]
p1 [3] = [x = 0, y = 0, z = .7]
p2[3] = [x = .26, y = 0, z = .3]

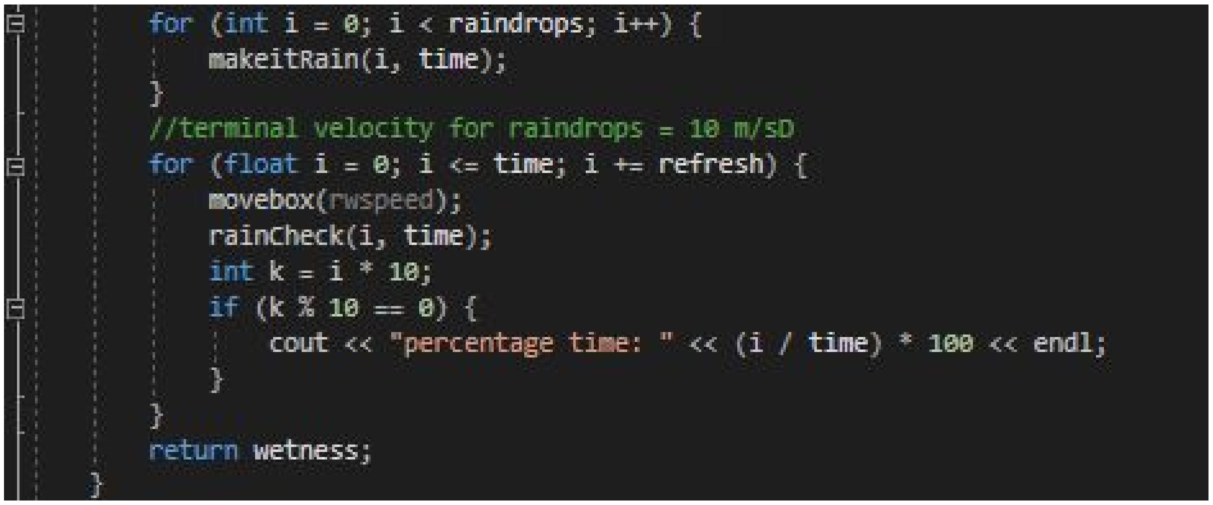
void makeitRain() : Making it Rain



```
void makeitRain(int i, float time) {
    //raindrop, droptop, smoking on cookie in a hotbox
    int tInt = time;
    vector<float> raindrop;
    raindrop.push_back((rand() % 100) + ((float)rand() / RAND_MAX));
    raindrop.push_back(10);
    raindrop.push_back((rand() % tInt) + ((float)rand() / RAND_MAX));
    raindrop.push_back(((float)rand() / RAND_MAX));
    rd.push_back(raindrop);
}
```

This function creates raindrops. The first thing we do is create a vector that is pushed a random x value between 0 and 100, a y value of 10 (arbitrary), a random trigger time between 0 and the time it takes to go from a to b in our simulation, and a random z value between 0 and 1. This vector is then pushed into our 2 dimensional vector that contains all of the raindrops. Calling this function inside of a for loop in our simulation function where i is <= raindrops creates all of the raindrops for our simulation.

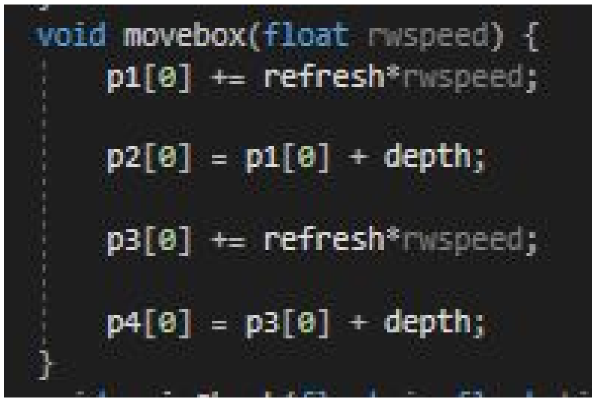
int simulation () Part II



```
        for (int i = 0; i < raindrops; i++) {
            makeitRain(i, time);
        }
        //terminal velocity for raindrops = 10 m/sD
        for (float i = 0; i <= time; i += refresh) {
            movebox(rwspeed);
            rainCheck(i, time);
            int k = i * 10;
            if (k % 10 == 0) {
                cout << "percentage time: " << (i / time) * 100 << endl;
            }
        }
        return wetness;
    }
```

The first for loop is used to create raindrops, and the second for loop is our simulation loop where i <= time and iterates with our simulation’s refresh rate.

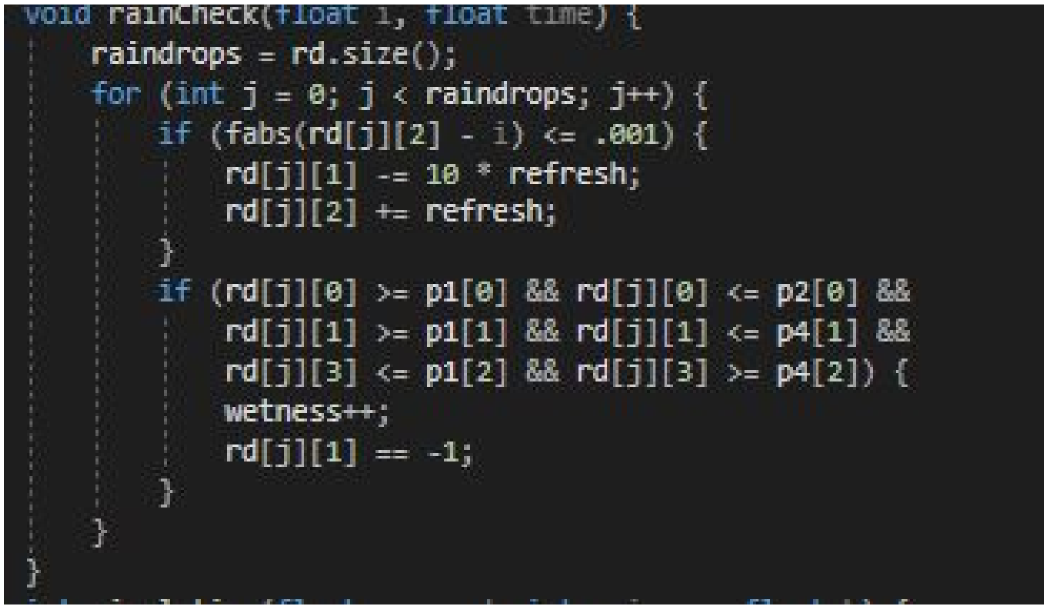
void movebox(): Why can't we just take the box and move it over there?



```
void movebox(float rwspeed) {
    p1[0] += refresh*rwspeed;

    p2[0] = p1[0] + depth;

    p3[0] += refresh*rwspeed;

    p4[0] = p3[0] + depth;
}
```

This function moves the x positions of the p arrays by the product of the refresh rate and speed. Doing this inside of a for loop will get p3[0] to 100 by time t.

void raincheck(): Can I get a raincheck on that?

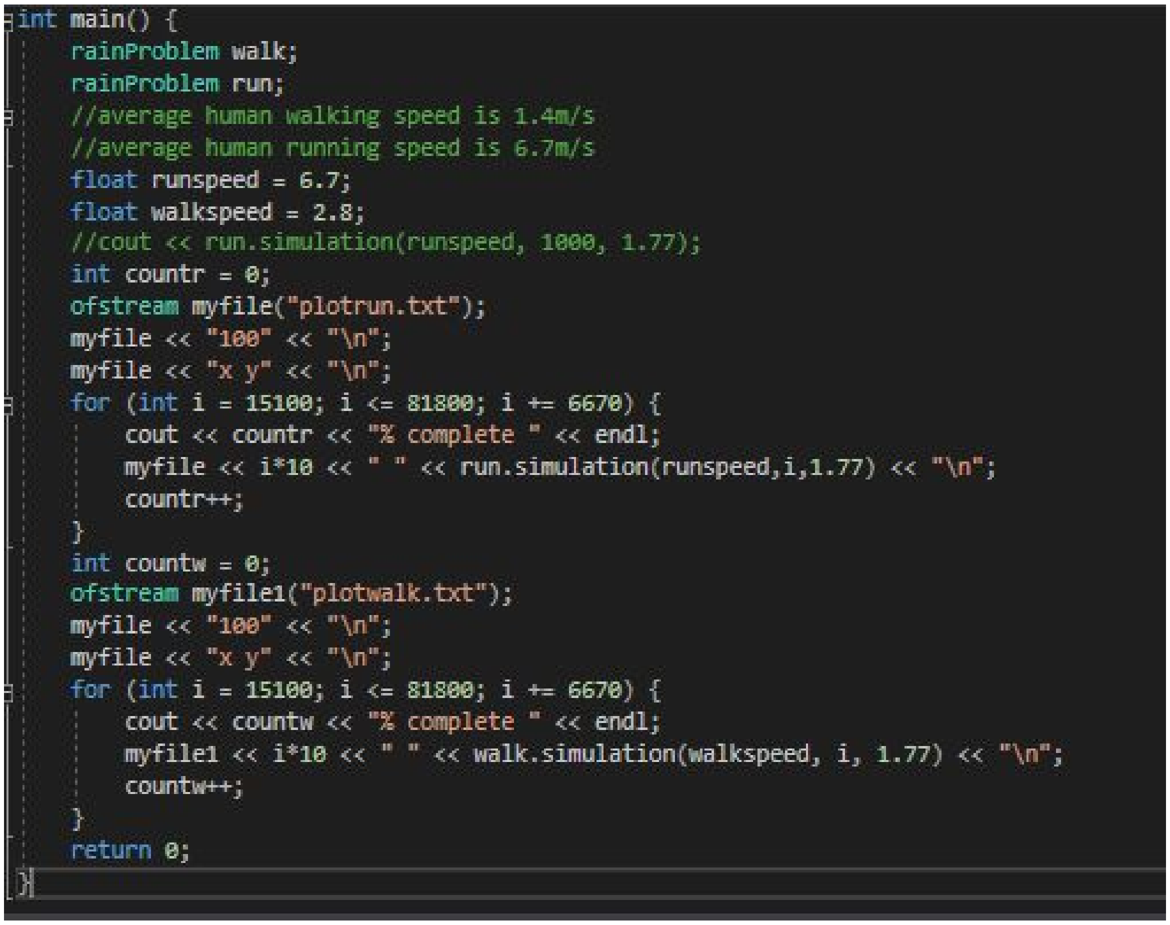


```
void rainCheck(float i, float time) {
    raindrops = rd.size();
    for (int j = 0; j < raindrops; j++) {
        if (fabs(rd[j][2] - i) <= .001) {
            rd[j][1] -= 10 * refresh;
            rd[j][2] += refresh;
        }
        if (rd[j][0] >= p1[0] && rd[j][0] <= p2[0] &&
            rd[j][1] >= p1[1] && rd[j][1] <= p4[1] &&
            rd[j][3] <= p1[2] && rd[j][3] >= p4[2]) {
            wetness++;
            rd[j][1] == -1;
        }
    }
}
```

This function sets the raindrop int to the size of rd (more computationally efficient). It then runs a for loop that loops through the raindrops in rd. If the jth raindrop's trigger time is about the same as the current time, it moves the raindrop's y value by the product of the raindrop's terminal velocity, and increases the trigger time so it keeps up with the current time.

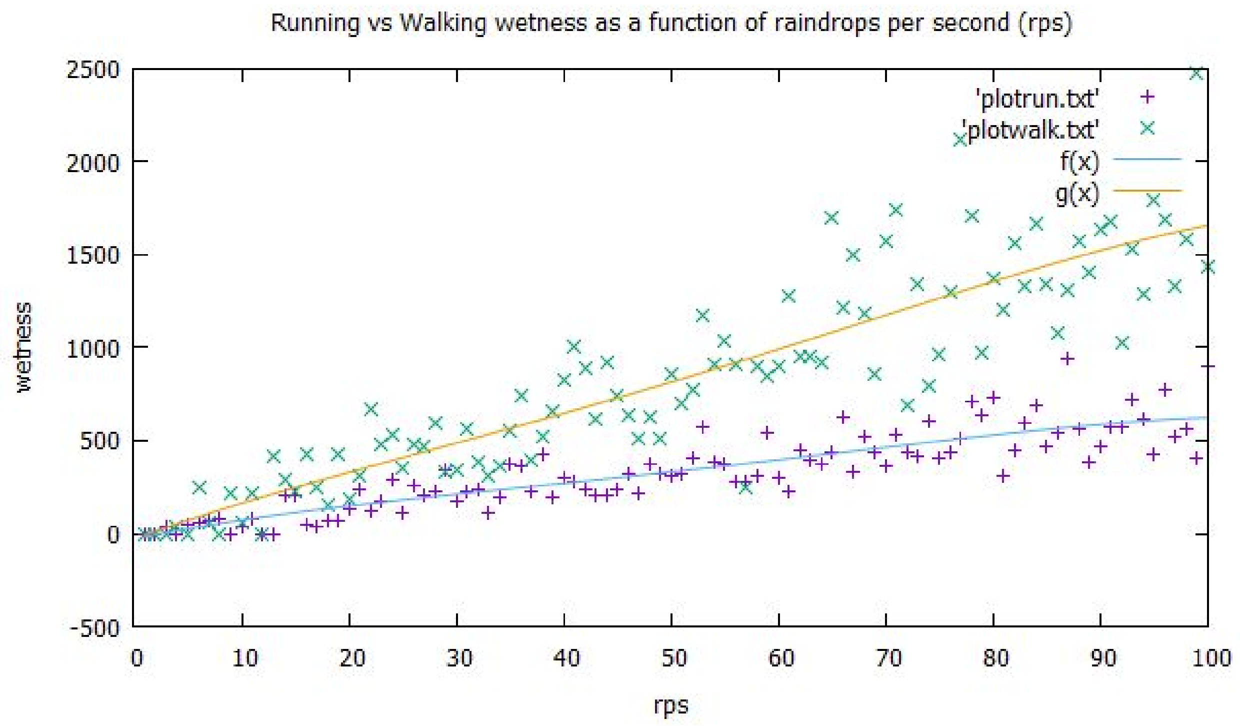
The second part of this function checks for collisions between the box and the raindrops in our simulation. If a raindrops x, y, and z positions are between the x, y, z positions of our box, we register it as a wetness point, and permanently set the raindrop's y value to -1 so we don't have to continue iterating it. After running the movebox and raincheck functions in our time for loop, we return the final wetness of our simulation, giving us our results.

Conclusion



```cpp
int main() {
    rainProblem walk;
    rainProblem run;
    //average human walking speed is 1.4m/s
    //average human running speed is 6.7m/s
    float runspeed = 6.7;
    float walkspeed = 2.8;
    //cout << run.simulation(runspeed, 1000, 1.77);
    int countr = 0;
    ofstream myfile("plotrun.txt");
    myfile << "100" << "\n";
    myfile << "x y" << "\n";
    for (int i = 15100; i <= 81800; i += 6670) {
        cout << countr << "% complete " << endl;
        myfile << i*10 << " " << run.simulation(runspeed,i,1.77) << "\n";
        countr++;
    }
    int countw = 0;
    ofstream myfile1("plotwalk.txt");
    myfile << "100" << "\n";
    myfile << "x y" << "\n";
    for (int i = 15100; i <= 81800; i += 6670) {
        cout << countw << "% complete " << endl;
        myfile1 << i*10 << " " << walk.simulation(walkspeed, i, 1.77) << "\n";
        countw++;
    }
    return 0;
}
```

Our int main() creates rainProblem objects, declares human running and walking speeds, and writes the results of our simulation with running and walking speeds to a file. We run the simulation many times inside of a for loop whereas i increases, the raindrops per second of our simulation increases. These get written to files called “plotrun.txt” and “plotwalk.txt”.



Above is a plot comparing the running and walking values of our simulation as rps increases, along with their curves of best it. As we can see, running is clearly better than walking in every instance. This result is rather disappointing, but coming to a conclusive answer is perhaps most satisfying thing about solving an ambiguous problem.