**Wiidrop**

**LAB 6**

**SECTION A**

**SUBMITTED BY:**

**Brian Reber**

**SUBMISSION DATE:**

**10/23/2009**

**Lab Problem**

Our goal is to use the sensors inside the Wiimote to tell us when to print out statements. We will have the program output something when it is waiting to be dropped, while it is falling, and when it hit. When it hits, it also needs to tell us how far it fell. In addition, we should tell how far it fell accounting for air resistance.

**Analysis**

The goal of this lab is to print out “I’m waiting” the first time through your loop while the Wiimote is waiting to be dropped. Then at a certain interval, we will print out periods. It will look like this:

I’m waiting…………………..

And it will print out a period when a certain decided upon amount of time has passed. Then when the Wiimote was dropped, it should do the same thing, just saying “Help me! I’m falling!!!!!!!!!!!!” with the exclamation points printing using the same idea as the periods. Then when it hits the ground, it will print out that it hit, and then calculate the distance it fell using the amount of time it was falling. Also, the program should exit when the plus button has been pressed.

**Inputs**:

There will be data coming in via the Wiimote and wiiwrap.

**Outputs**:

* When the Wiimote is waiting to be dropped, it will output:

“I’m waiting………………” with a varying amount of periods depending on the time it spends waiting.

* When the Wiimote is falling, it will output:

“Help me! I’m falling!!!!!!!!!!!!!!!!” with a varying amount of exclamation points depending on the time it spends falling.

* When the Wiimote hits the ground, it will output:

“Ouch! I fell \_\_.\_\_\_ meters in \_\_.\_\_\_ seconds. Accounting for air resistance, the fall was \_\_.\_\_\_ meters.” Where the values will be the actual values that it has done that.

**Design**

Necessary Formulas:

* Distance = (9.8 \* totalTime \* totalTime) / 2

With air resistance…

* Velocity = (prevVelocity) + (9.8 – magnitude(ax,ay,az)) \* (∆ time)
* Distance = (prevDistance) + velocity \* (∆ time)

Algorithm:

1. First off, we will create a function called “tolerance” so that when we need to check if the value is within a certain range, we can call this function instead of repeating it.
2. We will need multiple loops to make this program run most efficiently. The first loop will be the waiting loop. This loop will run as long as the magnitude is close to 1, meaning that the Wiimote is at rest. This loop will print out the necessary text, and each time after the first, it will print out a period.
3. We decided to reset the counter to 0 after the first loop so that we can tell the first time it goes into the second loop so we can print out the “I’m falling” the first time.
4. After the previous loop, we will need another loop for the falling stage. This one will run as long as the magnitude is close to 0, meaning that the Wiimote is experiencing zero g’s, and is therefore falling. This loop will print out the necessary text, and every time after the first it will print out an exclamation point.
5. We collect the time every time we go through the falling loop. That way we can find the change in time, and then use that to calculate the velocity and distance at each time. We are essentially summing the distance and velocity so that we can more accurately calculate the distance with air resistance.
6. Then after that loop, we will need to calculate the distance that the Wiimote fell, and print out the results and clear our variables so that it can run again.
7. Within each while statement, we will check whether the “B” button is pressed. If it is, we will exit out.

**Testing**

We ran into quite a few small problems when writing this program.

* Our first problem occurred when we copied the code from the previous lab into a new file. We decided to do this so that we didn’t have to retype all of the basic code that we needed. The problem was with the while loop around the whole program. It would not work. We ended up changing the variable in the while loop and it worked.
* Another problem happened when the periods and the exclamation points would print out in big bundles. The TA explained this issue to the whole group and told us to put a fflush(stdout) statement right after each printf statement we had. This allows the computer to print out each statement right away instead of combining statements.
* In part one, we originally decided to only have one while loop. We used an integer variable that would let us know where we were in the process (waiting, falling etc). This worked, but was probably inefficient. We then decided to have three while/do loops. The first was the overall loop that occurred whenever the plus button hadn’t been pressed. Inside this loop we had a do-while loop that ran while the ball was waiting (the magnitude was not close to 0 and plus hadn’t been pressed). Then after one of those two conditions had occurred, we went into the next one that happened while the ball was falling (the magnitude was not greater than 1.5 and the plus button hadn’t been pressed). We figured that this would be more efficient because it had less code to go through each time through a loop.
* Another problem we had was dealing with the part two. We were unsure of how to use the equations given to us. We tried many options, and eventually landed on the correct one. We tried ignoring the summation, but then realized that that was necessary.

**Comments**

I thought that this lab was challenging, but not impossible. We were able to figure it out, but it took a lot of thought and a collaborative effort on both partners. Some of our best ideas came from one person responding to a previous idea. We needed to think through the problem given to us, and try to figure out what it meant. I think that as we went we tried to make the code more efficient and not as “wordy.” I thought that this lab was really interesting and fun. I really enjoyed working on it and trying to figure my way through it.

**Implementation**

/\* Lab 6 Wiidrop Program

Brian Reber & Nathan Brinkman

\*/

#include <stdio.h>

#define TOL .15 //define tolerance for close to function

#define G 9.8

/\* Put your function prototypes here \*/

double mag(float x, float y, float z);

int minutes(float ms);

int seconds(float ms);

int leftover(float ms);

int close\_to(float tolerance, float point, float value);

int main(void) {

float ax, ay, az;

int run = 1;

int t,a,b,plus,home,minus,one,two,dup,ddown,dleft,dright;

int status = 0;

int stime = 0;

int etime = 0;

float tottime = 0;

float dist = 0, dist2 = 0;

int hit = 0;

double velocity = 0;

float t\_f = 0, t\_i = 0;

while (run) {

printf("I'm Waiting");

//Waiting loop

do

{

scanf("%d,%f,%f,%f,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d", &t, &ax, &ay, &az,

&a,&b,&plus,&home,&minus,&one,&two,&dup,&ddown,&dleft,&dright);

if (status % 10 == 0)

{

printf(".");

fflush(stdout);

}

status++;

if(plus)

{

run = 0;

printf("PLUS PRESSED, EXITING PROGRAM.");

}

} while (!(close\_to(TOL,0,mag(ax,ay,az))) && run);

//reset status to 0 so that we can tell the first time in the next loop

status = 0;

if (run)

{

printf("\n\n\tHELP! I'M FALLING");

stime = t;

//Falling loop

do

{

scanf("%d,%f,%f,%f,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d", &t, &ax, &ay, &az,

&a,&b,&plus,&home,&minus,&one,&two,&dup,&ddown,&dleft,&dright);

t\_f = t/1000.0;

if (status == 0)

t\_f = 0;

velocity = velocity + (G - mag(ax,ay,az))\*(t\_f-t\_i);

dist2 = dist2 + velocity \* (t\_f-t\_i);

t\_i = t/1000.0;

if (status % 10 == 0)

{

printf("!");

fflush(stdout);

}

status++;

if(plus)

{

run = 0;

printf("PLUS PRESSED, EXITING PROGRAM.");

}

} while (!(mag(ax,ay,az) > 1.5) && run);

}

if (run)

{

etime = t;

tottime = etime - stime;

tottime = tottime/1000.0;

dist = (G\*(tottime\*tottime))/2;

printf("\n\n\t\tOUCH! I fell %4.5lf meters in %4.3lf seconds.\n\n", dist, tottime);

printf("\t\tCompensating for air resistance, the fall was %4.5lf meters.\n", dist2);

printf("\t\tThis is %.2lf%c less than computed before.\n", ((1-(dist2/dist))\*100), 37);

}

tottime=0;

stime=0;

etime=0;

dist=0;

if(plus && run)

{

run = 0;

printf("PLUS PRESSED, EXITING PROGRAM.");

}

}

return 0;

}

double mag(float x, float y, float z)

{

return sqrt(x\*x + y\*y + z\*z);

}

int close\_to(float tolerance, float point, float value)

{

return (value < point+tolerance && value > point-tolerance);

}