Specification

Projectile flight

Suppose a cannonball is propelled straight into the air with a starting velocity v_0. The position of the ball after t seconds is $s(t) = -1/2gt^2 + v_0t$, where $g = 9.81m/s^2$ is the gravitational force of the earth. In our simulation, we will consider how the ball moves in very short time intervals Δt. In a short time interval the velocity v is nearly constant, and we can compute the distance the ball moves as $\Delta s = v\Delta t$. In our program, we will set

```
const double DELTA_T = 0.01;
```

and update the position by

```
s = s + v * DELTA_T;
```

The velocity changes constantly—in fact, it is reduced by the gravitational force of the earth. In a short time interval, $\Delta v = -g\Delta t$, we must keep the velocity updated as

```
v = v - g * DELTA_T;
```

In the next iteration the new velocity is used to update the distance. Now run the simulation until the cannonball falls back to the earth. Get the initial velocity as an input ($100m/sec$ is a good value). Update the position and velocity 100 times per second, but print out the position only every full second. Also printout the values from the exact formula $s(t) = -1/2gt^2 + v_0t$ for comparison.

The first part of the assignment was submitted to Mimir already (5 points)

Part 2: Use FLTK to animate the flight of the cannonball based on the positions we calculated.

The user will enter an initial velocity in the input box, then press the button to see the cannonball change positions and eventually fall back to Earth. (5 points)

We will learn the techniques this week...including how to capture the flight in a video!

Document the lab as usual...

Upload and submit: **pdf**, **tgz**, and **avi** files

Analysis

Inputs The initial volecity $v_0, \Delta t = 0.01, g = 9.81$

Process Calculate the distance s (position of the cannonball) from input v_0, v,and update it. where at the first time,$v = v_0$and then we can enter the loop where $v = v - g * \Delta t, s = s + v * \Delta t$ will be process every 0.01 second.

(And at the same time,we need to use fltk to make a window where the user will enter an initial velocity in the input box, then press the button to see the cannonball change positions and eventually fall back to Earth.)

Outputs The distance will be translated into the position of the ball by using the functions representing width and height as x and y in coordinate system.That's how we get our output.

Design

1. Define constants and variables

   The graviational constant is 9.81 m/sec

   v is velovity, s is position, t is time

   Delta time is constant intercal = 0.01

2. Get initial velovity from user

3. set v to initial velovity

4. Display column hraders

5. Reapte until cannonball hits the earth

   (a) display each row each hundredth iteration(seconds, incrementslly calculated position, position calculated using formula)

   (because dispay seconds but time interval is 0.01 seconds)

   (b) Update new position

   (c) Update new velocity

Implementation

```
// cs102 lab
#include <iostream>
#include "labgui.h"
int main()
{
        make_window()->show();
        return Fl::run();
}
```

```
// generated by Fast Light User Interface Designer (fluid) version 1.0304

#include "labgui.h"

Fl_Box *field=(Fl_Box *)0;

Fl_Box *bb=(Fl_Box *)0;

static void cb_Go(Fl_Button*, void*) {
  std::cout << "ouch" <<std::endl;
Fl::add_timeout(1.0, move);
}

Fl_Value_Input *initial_v=(Fl_Value_Input *)0;
```

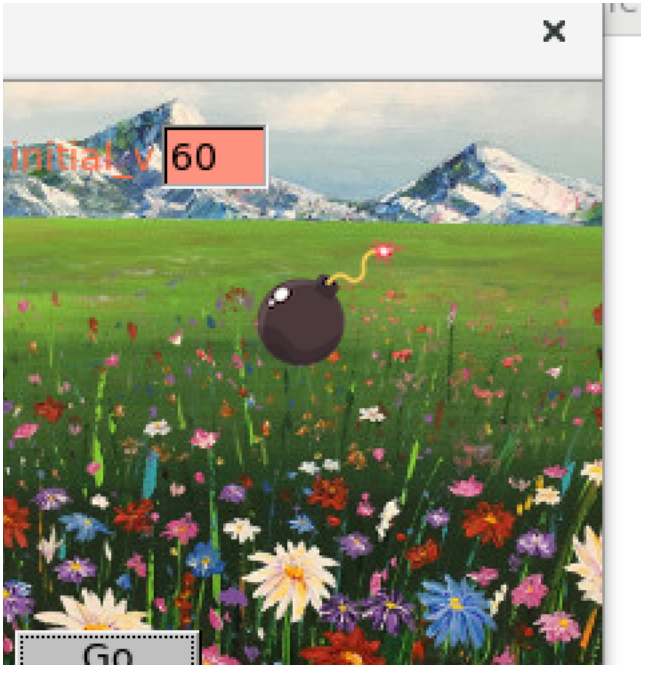
```
Fl_Double_Window* make_window() {
  Fl_Double_Window* w;
  { Fl_Double_Window* o = new Fl_Double_Window(225, 225);
    w = o; if (w) {/* empty */}
    { field = new Fl_Box(0, 0, 225, 225);
      Fl_PNG_Image* bg = new Fl_PNG_Image("field.png");
      field->image( bg );
    } // Fl_Box* field
    { bb = new Fl_Box(102, 205, 10, 20);
      Fl_PNG_Image* bi = new Fl_PNG_Image("l.png");
      bb->image(bi);
    } // Fl_Box* bb
    { Fl_Button* o = new Fl_Button(5, 205, 70, 20, "Go");
      o->callback((Fl_Callback*)cb_Go);
    } // Fl_Button* o
    { initial_v = new Fl_Value_Input(60, 16, 40, 24, "initial_v");
      initial_v->color((Fl_Color)172);
      initial_v->labelcolor((Fl_Color)131);
    } // Fl_Value_Input* initial_v
    o->end();
  } // Fl_Double_Window* o
  return w;
}

void move(void*) {
```

```
  // initialize these values only the first time
  static double dt = 0.01;
  static double x = bb->parent()->w()/2 - bb->w()/2;     //center
  static double y = bb->parent()->h() -bb->h();    //bottom
  static double s=0;
  static double v0 = initial_v->value();//get input
  static double v=v0;
  static const double g=-9.81;// it is easier to calculate when we use the ne
  v+=g*dt;//the reason wht it is plus g is that we used g=-9.81.
  s+=v*dt;

  y = bb->parent()->h()-s-bb->h();//translate s into y.


  bb->position(x,y);// the position in the x-y coordinary system
  bb->parent()->redraw();
  std::cout<<"distance: "<<s <<std::endl;
  std::cout<<"now the velocity is : "<<v<<std::endl;

  if(s>0)//to control it to hit the earth.
  {
  Fl::repeat_timeout(0.01, move);
  }
}

// generated by Fast Light User Interface Designer (fluid) version 1.0304
```

```
#ifndef labgui_h
#define labgui_h
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_PNG_Image.H>
extern Fl_Box *field;
extern Fl_Box *bb;
#include <FL/Fl_Button.H>
#include <iostream>
#include <FL/Fl_Value_Input.H>
extern Fl_Value_Input *initial_v;
Fl_Double_Window* make_window();
void move(void*);
#endif
```

Test

Testcase (screen shoot)



As we can see in the picture, my program is running successfully.The background, the bomb, and our input box appear in the corrct position.

Testcase (movie)

the name of my Avi files are outputc.avi ,outputc2.avi .

As we can see in the avi files, my program is running successfully.The position of the bomb update correctly, and hit the earth at last.At the same time, I have the output of the velocity and distance in the terminal , which looks nice.