CSE 232 Fall 2017

Programming Project #8

Assignment Overview

This project focuses on the use of structs and struct methods. It is worth 60 points (6% of your overall grade). It is due Monday 11/06 before midnight

The Problem

One of the problems one runs into when building games or videos is dealing with the physics of motion in the video. We are going to look at a small subset of physics, the elastic collision of a 2D ball with a wall and with another ball. We don't have the equipment to make the pictures that go with it, but we can at least do the physics.

Vectors, the math kind!

You did a lot of this in lab09, we are going to build on that to help us with our problem.

The TwoD struct

The TwoD struct represents a math-vector in two dimensions. We can interpret the two values in the vector in one of two ways for our problem:

- as an x,y position in 2D space
- as a x-velocity and y-velocity in 2D space

same struct can be used for either.

We provide the file `proj08_twod.h` which declares the following structure.

```
struct TwoD{
  double x = 0;
  double y = 0;

  TwoD()=default;
  TwoD(double xval, double yval);

  string to_string();
  TwoD diff(TwoD);
  TwoD mult(double);
  double dot_product(TwoD);
  double magnitude_squared();
};
```

Each variable declared of type `TwoD` will contain two data members: `x` and `y`. It also contains the declaration of two constructors (one already defined as a default) and 5 function members. You need to implement the non-default constructor and the 5 function members:

- `to_string` : a TwoD method. Returns a string of the form `"(x.00, y.00)"` where x and y are the values stored in the struct. If you use a stream, then the stream manipulators fixed and setprecision(2) could be used. For example `(1.00, 23.20)` See the Mimir tests for details.
- `diff` : a TwoD method. The difference between the two vectors. The calling `TwoD` element is first, the argument `TwoD` second. Returns a new `TwoD`. You basically did this in lab09

- `mult`: a TwoD method. Multiply both elements of a `TwoD` by the provided `double`. Returns a new `TwoD`. Also done in lab09
- `dot_product` : a TwoD method. You can look this up, but for two `TwoDs` `v1` and `v2`, you return a double which is: `(v1.x * v2.x) + (v1.y * v2.y) .` Also done in lab09
- `magnitude_squared` : a TwoD method. You can look this up, but for a `TwoD` `v1`, you return a double which is:
  `(v1.x * v1.x) + (v1.y * v1.y)` Similar to work in lab09

Assignment 1
Write a file `proj08_twod.cpp` that provides the definitions for the declarations of `proj08_twod.h`.

The Ball struct

The Ball struct represents a 2D ball. The structure is provided in `proj08_ball.h` and contains the structure declarations:

```
struct Ball{
  double mass = 0;
  TwoD coords;
  TwoD velocity;
  double radius = 0;

  Ball()=default;
  Ball(double m, TwoD pos, TwoD speed, double r);

  string to_string();
  bool contact_ball(Ball);
  bool contact_wall(long xdim, long ydim);
  TwoD update_velocity_ball(Ball);
  TwoD update_velocity_wall(long xdim, long ydim);
};
```

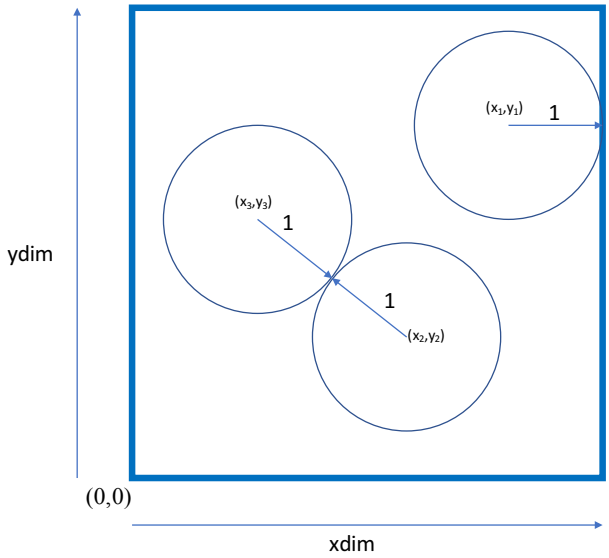
Each variable of type Ball will have 4 data members:

- a mass (double)
- a `TwoD` coords (it's position in 2D space)
- a `TwoD` velocity (the x-velocity and y-velocity of the moving ball)
- a radius (double)

It has two constructors and 5 function members. You have to implement the non-default constructor and the 5 function members:

- `to_string` : a Ball method. Returns string of the form:
  `1.00, (1.00, 1.00), (1.00, 1.00), 1.00`
  - first is `mass`, second is `coords`, third is `velocity`, fourth is `radius`

The rest will take a little more work. Let's look at some geometry



Ignoring the mass for the moment, let's just talk about contact:

- a ball is in contact with another ball when the distance between their two centers is equal to or less than the sum of the two radii of the balls
- a ball is in contact with a wall if the distance between the center and the wall is less than or equal to the balls radius. We assume that we the balls are always contained in a rectangular box whose lower left coordinate is (0,0), and that we know the extent of the box in both x and y. Let's talk about the x-dimension. A ball is in contact with the wall if:
    - if the coords.x plus the radius is greater than xdim
    - if the coords.x minus the radius is less than 0.

  a similar calculation can be made for the y-dimension

So …

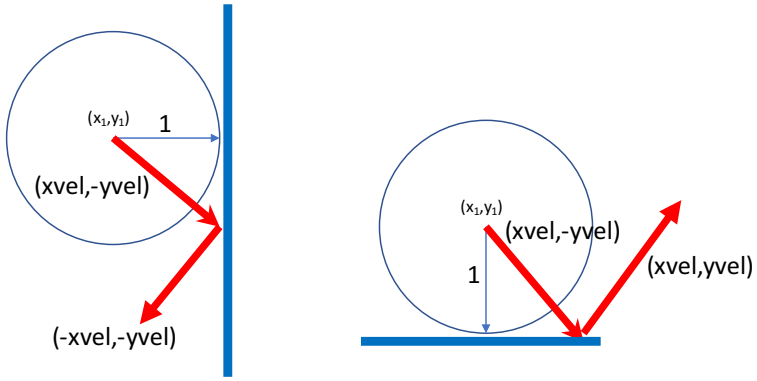
- `contact_ball` : a Ball method. Takes another `Ball` as an argument and returns `true` if the distance between their two centers is less than the sum of their two radii. False otherwise.
- `contact_wall` : a Ball method. Takes the `x-dim` and `y-dim` (as shown above) and returns `true` if any contact occurs between the ball and the wall as described above.

Special Note

It can be tricky to do exact comparisons of floating point numbers. Because calculations end up with approximate, but very small, differences, exact comparison, i.e. equal, can be difficult. What we promise is that in our test cases the ball/wall comparison will be clearly overlapping (unequivocal in whether the ball/wall is touching). It may not be very "physical" but it will be clear.

Change in Velocity

- `update_velocity_wall`. A Ball method. Takes the xdim and ydim of the box and returns a `TwoD`, the change of velocity. Look at the diagram below



The rule is pretty simple. For the dimension that has contact (x or y), the sign of the associated x-velocity or y-velocity component is changed.

  - In the first example, the ball is moving at (xvel, -yvel), that is to the right and down. Contact is in x so the associated x-velocity sign is changed. It's now moving to the left and down.
  - In the second the ball is moving at (xvel, -yvel) (to the right and down). Contact is in y so the associated y-velocity sign is changed. It's now moving at (xvel, yvel) (to the right and up)

- `update_velocity_ball` : A Ball method. If two balls are in contact, what is the change in velocity? This is the hardest one to write. Let's break it down. (from https://en.wikipedia.org/wiki/Elastic_collision, bottom of the page)

  First here is the vector equation. Relax, we have built all the tools to deal with this!

$$\vec{v_1'} = \vec{v_1} - \frac{2*mass_2}{mass_1+mass_2} * \frac{(\vec{v_1}-\vec{v_2})\cdot(\vec{x_1}-\vec{x_2})}{\|\vec{x_1}-\vec{x_2}\|^2} * (\vec{x_1}-\vec{x_2})$$

  The little arrows above the variables indicate that we are doing vector-math operations. For this equation, we need:

  - vector difference
  - dot product, the dot (•) in the second fraction
  - magnitude_squared, the $\|\ \ \|^2$ in the second fraction
  - multiplication, vector * value

  But we have all that in our `TwoD` struct!!! You just have to break it down. The $\vec{x}$ represents the position `TwoD` of the two balls, the $\vec{v}$ the velocity `TwoD` of the two balls. We can calculate the updated velocity, $\vec{v_1'}$ of the first ball by applying this equation. This is what `update_velocity_ball` does, return the updated `TwoD` vector of the change in velocity.

Assignment 2

Write a file `proj08_ball.cpp` that provides the definitions for the declarations of `proj08_ball.h` Since you use `TwoD` in `Ball`, you should include `proj08_twod.h in` your `Ball` code.

Test Cases
Test cases are provided in Mimir as always.

Assignment Notes
1. You are given the following files in the downloaded Student Started code:
    a. `proj08_main.cpp` This is a file you can modify as you like to test your code.
    b. `proj08_twod.h` and `proj08_ball.h` These will be used as is in Mimir testing.
2. You will write and turn in ***both*** `proj08_twod.cpp` and `proj08_ball.cpp`
    a. There's a video from Week 4 on how to upload to Mimir. You could test locally and upload using that approach.

Deliverables
1. Remember to include your section, the date, project number and comments.
2. Please be sure to use the specified full directory and file name, i.e. `proj08/proj08_twod.cpp` and `proj08/proj08_ball.cpp,` that is both files in the same directory
3. Submit to Mimir.