README.md

FooBall - github.com/Googe14/FooBall

St Philips College - Q1 2019 Year 11 Digital Technologies assignment

The source code for this project can be found in FooBall/src Starting at FooBall_App.class with multiple compiled versions at different points in development.

This project is very heavily commented for your convenience

Storyboard/Roadmap

The objective of this application is to recreate the classic Doccy Jo bouncing balls tutorial, but expand by instead of having a game, make more of a sanbox application where users can mess with the physics of the balls.

Objectives:

- Create frame with balls
- Have a side bar for settings
- Create a number of balls that will bounce and interact with each other
- Have a number of settings for people to play around with

Interactions between balls

- Collision detection between circles
- Seperation method to move circles out of each other if they go into each other
- Collision physics (2D elastic collisions between circles)

Possible Extra features:

- Add pause and continue functionality
- Select individual balls while paused - edit individual ball's settings'
- ~~Dynamically alter number of balls~~
- Multiple game modes

Expected input

Changing of settings:

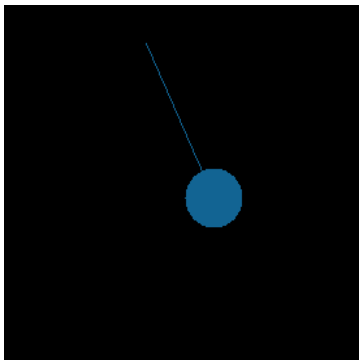
- Gravity (slider)
- Mouse Strength (slider)
- Air resistance (slider)
- Mouse mode (radio button)
- Type of collisions (check boxes)
- Scope of mouse effects (radio buttons)
- Number of balls (spinner)

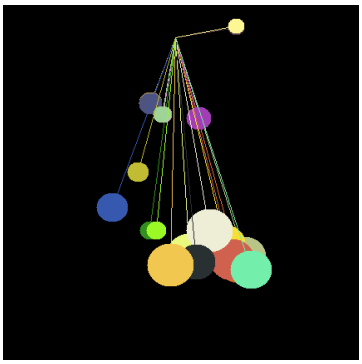
Mouse clicks:

- Attract balls
- Repel balls
- Grab balls
- Slingshot balls

Expected output

Balls will interact with each other accordingly (bouncing, colliding etc) whilst any mouse input with the user inside the black panel will also affect the balls depending on what mode the user has selected.





Design Goals/Structure

- Create JFrame/window to hole contents
- Create Panel to display graphics in Frame
- Write game loop to time events of program
- Create objects to hold data for objects
- Create boundaries and write physics to redirect balls at boundaries
- Write collision detection between balls
- Write physics for colliding balls to bounce off each other

Boundaries of objects

Each PhysicsObj object stores two integers indicating the distance from the origin (top left, [0,0]) which marks it's boundary. i.e. how far it is allowed to travel before it bounces.

```
//Bounce the ball
//Bounce on Right side
if(x >= xBounds-size/2) {
        xVel *= -1;
        x = xBounds-size/2;
}
//left side
if(x <= size/2) {
        xVel *= -1;
        x = size/2;
}
//bottom
if(y >= yBounds-size/2) {
        yVel *= -1;
        y = yBounds-size/2;
}
//top
if(y <= size/2) {
        yVel *= -1;
        y = size/2;
}
```

[FooBall/src/FooBall_PhysicsObj.java move()]

x and y representing the coordinates of the center of the ball, size being the diameter, and x/yBounds being the upper limit of their movement.

This method simply checks if the position of the ball +/- half it's size (to get the position of the side of it, so the centerpoint + the radius) is equal to or has gone beyond the bounds.

If it has, it reverses the velocity of the ball (x or y depending on which side it collided with) and sets the position to be right on the bounds, as for the ball not to get stuck outside of the bounds.

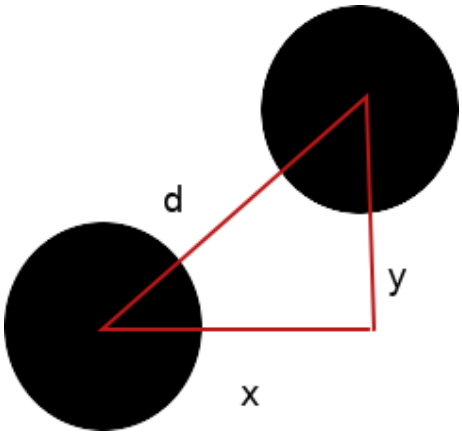
Collision detection

At first thought, collision detection between balls seemed intimidating, compared to collision detection between boxes where you just check if the edges are within the domain and range of the other box (later found out to be called Axis-Aligned Bounding Boxes collision detection) However, this was a very simple problem, as the distance between two points can be found using Pythagoras, and the center of two balls can just be treated as two points.

```
//Get x positions of balls
int x1 = ball1.getX();
int x2 = ball2.getX();

//Get y positions of balls
int y1 = ball1.getY();
int y2 = ball2.getY();
//Get the distance between those coordinates
int distance = (int)Math.sqrt((x2-x1)*(x2-x1)+ (y2-y1)*(y2-y1));
//Check if they are touching
if(distance <= ball1.getRadius() + ball2.getRadius()) {

}
```



[FooBall/src/FooBall_Physics.java compareBalls()]

The x and y displacements between the two balls are found, these lengths can be treated as the two shorter sides of a right-angled triangle. With these distances, we can treat the actual distance between the balls as the hypoternuse of the triangle, which can be found using Pythagoras' theorem.

This resultant length is compared to the length of the two radi summed, and if the resultant length is less than the sum of the radi, we know that the two balls are intersecting, if it is equal, the balls are touching, and if it is greater, the balls are not in collision and the program can continue.

Ball separation

In a program like this, with a very simple physics engine, it is very possible, in-fact, likely, that when two balls collide they may have moved more than a single pixel at one moment and will go inside each other. Due to this intersection, when two balls bounce and they are inside each other, they may not move far enough to go completely outside of each other, especially with air resistance, and will immediately collide again, which is not realistic, and thus, not what we want. To fix this, we need a method that will take two balls that are inside each other and reposition them enough that they will be touching, but no longer intersecting.

```
//Get ball positions
        int x1 = ball1.getX();
        int y1 = ball1.getY();

        int x2 = target.getX();
        int y2 = target.getY();

        //Get ball sizes
        int r1 = ball1.getRadius();
        int r2 = target.getRadius();

        //Create vector for ball point differences
        float vx = x2-x1;
        float vy = y2-y1;

        //Get original distance apart of balls
        int px = (int)vx;
        int py = (int)vy;

        //Get target distance apart of balls and set it as difference
        int td = r1+r2;
        td -= (int)Math.sqrt(px*px + py*py) - 2;
        if(vx != 0 || vy != 0) {
                //Get length of vector
                float vl = (float)Math.sqrt((vx*vx + vy*vy));
                //Turn into unit vector
                vx = vx/vl;
                vy = vy/vl;
        }

        //Move balls out of each other
```

```
target.setPos(target.getX()+(vx*td/2), target.getY()+(vy*td/2));
ball1.setPos(ball1.getX()-(vx*td/2), ball1.getY()-(vy*td/2));
```

[FooBall/src/FooBall_Physics.java separateBalls()]

This is done by creating a vector between the position of the two balls, collecting the two radi of the balls and sum them together to get the distance between the two balls that we want to aim for, and also finding the distance between the two balls (refer to previous section about collision detection) to see how much we need to adjust them by. This offset distance is found by subtracting the actual distance from the target distance.

When two balls intersect, we want them to separate directly opposite to each other along the vector line between them, which is why we got the vector at the start, but for this direction to be useable to us, we need to turn it into a unit vector so we can scale it to the distance we want.

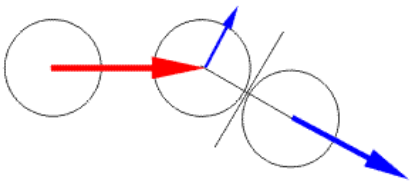
Finally, now that we have the distance we need to add between them and the scalable vector along which we will add this distance, we multiply the unit vector we have by half of the distance we need to add, and then add/subtract respectively that final amount to the positions of the balls. We are only using half of the target distance because we are adding it to *both* balls, thus we are adding two halves to get the entire target distance.

Ball bounces

This is the main attraction of the program now, the realistic interactions between balls in terms of how they bounce.

This part of the project took the most time, effort and research of this entire project as it does not occur to me that the mathematics behind how two balls bounce is a common knowledge in year 11. But eventually it did work.

When two balls collide with each other, two axis of movement are created. These axis of movement are a tangent and normal between the *position* of the two balls when they touch. The axis of movement are **not** dependant on their velocities. At the point of the collision, the ball that has been hit will react by moving in the normal direction opposite to the side the collision occured, however, the ball that initiated the collision will move in the tangental direction of these axis. With the directions of movement sorted, the magnitude of these two directional vectors are actually the components of the original velocity vector in the direction of the reaction axis. If this explanation was a bit hard to follow, there is a nice gif to illustrate this reaction.



```
int x1 = ball1.getX();
int y1 = ball1.getY();

int x2 = target.getX();
int y2 = target.getY();
//Get ball masses
float m1 = ball1.getMass();
float m2 = target.getMass();
//Get ball velocities
float v1x = ball1.getVelX();
float v1y = ball1.getVelY();

float v2x = target.getVelX();
float v2y = target.getVelY();
//Get distance between balls
float balld = (float) Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
//Create normal with ball positions and distance
float nx = (x2-x1) / balld;
float ny = (y2-y1) / balld;
//Create tangent from normal
float tx = -ny;
float ty = nx;
```

```
            //Dot products of tangent
            float dpTan1 = v1x * tx + v1y * ty;
            float dpTan2 = v2x * tx + v2y * ty;

            //Dot products of normal
            float dpNorm1 = v1x * nx + v1y * ny;
            float dpNorm2 = v2x * nx + v2y * ny;
            //Get conserved eneragy scalar
            float fm1 = (dpNorm1 * (m1-m2) + 2.0f * m2      * dpNorm2) / (m1+m2);
            float fm2 = (dpNorm2 * (m2-m1) + 2.0f * m1      * dpNorm1) / (m1+m2);

            //Bounce balls
            //Transfer force to other balls
            ball1.setVel(tx*dpTan1 + nx * fm1, ty*dpTan1 + nx * fm1);
            target.setVel(tx*dpTan2 + nx * fm2, ty*dpTan2 + nx * fm2);
```

[FooBall/src/FooBall_Physics.java collideBalls()]

To start with, some useful data is collected from the balls such as the positions, velocities and masses. A normal is found between the balls and a tangent is obtained from this normal by swapping the x and y and making one negative. The dot products are then taken from the vectors of velocity and the tangent/normal, this is done to get a measure of how much of the velocity vectors are in the direction of the tangent/normal axis. Finally, the magnitude of the new velocities are scaled according to preservation of momentum, and all these numbers come together when setting their velocity to these new values.

Objects

In the FooBall source folder there are seven Files, each containing one of the seven classes that work together to make this program work.

This consists of:

FooBall_App.java

This file contains the root class with the main method that the program will start from, in here, we only initialize a couple variables that will be passed to the rest of the program for it to run, like the screen size and the theme of the window. After that, it creates the JFrame and passes flow of the program to there.

FooBall_Frame.java

This class has two main sections, the first is to create and initialize the two panels that will be going in it (the graphics panel and the user control panel). Once it has created, initialized and added the two panels to itself, it send some listeners to the user control panel so that it can handle anything that happens when a button is pressed or a slider is slid etc and then pass the information on to the graphics panel. Once the panels are initialized, flow of the program is handed to the the main panel.

FooBall_UserPanel.java

This class has a lot of code but not much content, the majority of this class is simply the creation and placing all the buttons on itself and then adding the event handlers from the Frame to the buttons, slider, radios etc.
Alongside that, this class has two methods that are used to determine the values and positions of the sliders on a logarithmic scale, as sliders in java do not have in-built functionality to have a logarithmic scale.

FooBall_GamePanel.java

This class is perhaps the most complex class in this program. To start with, it has a few methods that are used to implement multithreading, this is necessary as java does not like to repaint if it is called multiple times in quick succession if it is on the main thread. The solution to this is to split it into a separate thread and run the loop from there.
After the multithreading implementation, this class has the game loop, it is possible to split the game loop into it's own class using lambdas to pass certain actions to it, however, I do not have experience with lambda expressions in java and so it is just inside the GamePanel class. Another possiblity to tidy this class up would also be to have the game loop call an abstract method and have a child class that implements this class with the abstract method, however, this is not a big deal and it serves the purpose it has in this form, so I have not bothered to do this.

After the game loop for this program, we come across the paintComponent method which is used to render graphics to the panel in which it is implemented. This method is never called directly, however it is invoked by another method when repaint() is called. The paintComponent method is passed a Graphics object which is then used to render graphics with certain methods such as drawOval and fillRect which are looped and passed the details of the balls to draw all the balls on the screen.

After the paintComponent, we have a number of methods and some objects which are used for manipulation of the balls, such as if the velocities of the balls are illustrated through a temperature, if there will be collisions between the balls etc. There are also a number of methods below again which are used to manipulate these, certain methods that are called by the Frame when it recieves an event from the user panel to update the physics of the balls and some of the other variables that were mentioned before. Alongside certain methods which interact with the balls themselves, for example, changing the number of balls and generating a temperature colour depending on their velocities.

Finally, apart from all those, we have the method which is used to update the state of the game. This method loops through all the existing balls and tell them to move, collide, separate and bounce among each other, if necessary (according to the variables mentioned earlier).

FooBall_PhysicsObj.java

This class serves as an interface for a possible number of different physics objects, as well as being a fully useable physics object itself. This class mostly just stores a bunch of data like its position, velocity, size, colour, mass, density (unused but exists for possibility of future expansion) and acceleration. The majority of this class otherwise are setters and getters for these variables, a method to generate a random new colour for itself, and an unused initialization method.
This class does also contain a method to move or change it's data depending on the current state of that data, change its position depending on velocity, change it's velocity depending on it's acceleration, change it's velocity due to gravity etc, and even bounce depending on what bounds it has set.

FooBall_Ball.java

This class is fairly minimal, and yet the most used object in this program. This class implements the PhysicsObj class, so it has all the properties of that class and is fully functional as a PhysicsObj object, however it has another variable, the radius, and a couple changes to the setter for the PhysicsObj size which is to ensure the radius is included and will always be accurate to the size. It also has a method to generate an acceleration towards a point depending on the distance from it on each axis.
The only other difference is that it also has an id variable which is/was used to ensure that a ball is not checked against itself for collisions and bounces etc.

FooBall_Physics.java

The final class in this program is the physics class which is used for, well, the physics. This object is created in the GamePanel class and is used to calculate things such as collisions, bounces and comparisons. It has a method which check to make sure there is space available for a ball in a certain position, the method to separate two balls that are touching, the method the *check* if those two balls are touching, a method which checks every ball in an ArrayList to each other to then check collisions, separationg and bounces. And finally, it has the method which is used to bounces two balls.
This is all done from FooBall_Ball objects that are passed to the Physics object from GamePanel.