Adding conditional statements to the game (Part 2)

Additional Resource

This document complements the Steps provided in Week 3 for adding conditional statements.

The game wouldn't be a game if there were no collisions between the objects in it, so you will need to learn how to detect collisions and react to them appropriately. For this you will be using some maths. Don't worry we will show you step-by-step how to do it.

Let's observe two balls that have collided (here R1 and R2 are the radiuses of balls)

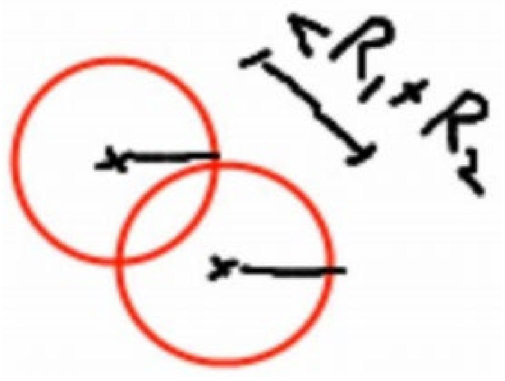

Figure 1: Two balls in collision

You see that there should be a collision if the distance between the centres of the balls (let's say 'c') is smaller than the combined length of the two radiuses (R_1 and R_2).

That is when

$c< R_1+R_2$.

Likewise there should not be a collision if

$c>R_1+R_2$

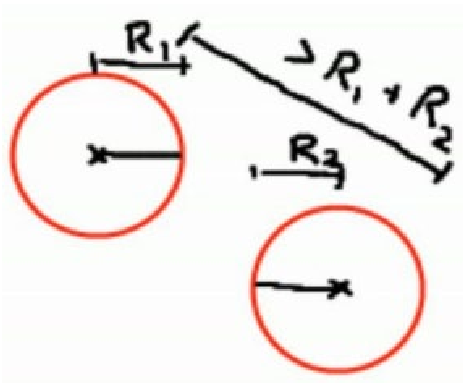

Figure 2: Two balls not in collision

So at the collision point

$c=R_1+R_2$

Equation 1: Collision condition

$c=R_1+R_2$

Pythagoras Theorem

Pythagoras Theorem teaches us that for any rectangle triangle: $c^2 = a^2 + b^2$

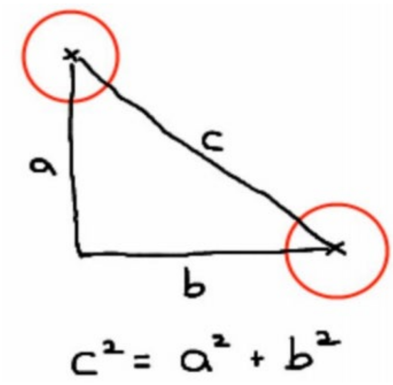

Figure 3: Pythagoras Theorem

You can learn more about Pythagoras' Theorem with Math Is Fun.

Applying Pythagoras Theorem

We use this theorem to calculate the distance, 'c', between the two centres of balls that are likely to collide.

It was shown that a collision should happen if:

$C<=R_1+R_2$

Note: $c =R_1+R_2$ is the moment of collision. You need to check for $c<=R_1+R_2$ in the condition because there are update intervals.

Using basic algebra (squaring both sides of the equation) it can also be said that:

$$C^2<=(R_1+R_2)^2$$

By using this equation you can avoid taking the square root, which can be slow on a computer compared with performing a multiplication.

Collision

We check collision detection in the **updateGame** method.

There can only be a collision between the ball and the paddle if the ball is moving towards the paddle. So we first check for a positive speed in Y direction.

```
if(mBallSpeedY > 0)
```

Note: this is not exactly correct; but correct most of the time. This simplifies logic in the game.

We need to calculate the distance between the ball and the paddle and we need a variable to store it. The variable distanceBetweenBallAndPaddle referrers to c2 in the collision equation. This variable should be defined before the if condition to check the Y directional speed of the ball.

```
float distanceBetweenBallAndPaddle;
```

Then let us calculate the distance between the ball and the paddle.

```
distanceBetweenBallAndPaddle = (mPaddleX - mBallX) * (mPaddleX - mBallX) +
(mCanvasHeight - mBallY) * (mCanvasHeight - mBallY);
```

If the minimum distance between the ball and the paddle is less than or equal to actual distance between the ball and the paddle, we have a collision.

Define mMinDistanceBetweenBallAndPaddle as a float (at the top where other variables such as mBallX and mBallY are defined).

```
private float mMinDistanceBetweenBallAndPaddle = 0;
```

Calculate the minimum distance using the radiuses of the two balls. This needs to be in the **setupBeginning**.

```
mMinDistanceBetweenBallAndPaddle = (mPaddle.getWidth() / 2 + mBall.getWidth()
/ 2) * (mPaddle.getWidth() / 2 + mBall.getWidth() / 2);
```

Note: here again mMinDistanceBetweenBallAndPaddle is the square of the minimum distance between ball and the paddle.

For the moment, if there is a collision we will change the speed of the ball to -200.

At this point the added code should look something similar to:

```
float distanceBetweenBallAndPaddle;

//If the ball moves down on the screen perform potential paddle collision

if(mBallSpeedY > 0) {
      distanceBetweenBallAndPaddle = (mPaddleX - mBallX) * (mPaddleX -
mBallX) + (mCanvasHeight - mBallY) * (mCanvasHeight - mBallY);


      //Check if the actual distance is lower than the allowed => collision

      if (mMinDistanceBetweenBallAndPaddle >= distanceBetweenBallAndPaddle) {

                            mBallSpeedY = -200;

                      }

      }
```

Now let us try to add logic that would make the collision more like a real life collision. But we will use assumptions to make out life easy.

When the collision happens you can cheat a bit and create this physical “rule”:

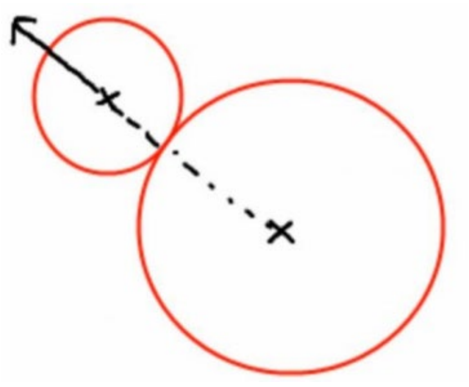

Figure 4: Collision point

The small ball will go (ricochet, bounce off) in a new direction, in a straight line away from the object it has collided with, as if the ball hit the object without an angle. This is easier than real physics, because the directions can be set by taking the centrum of the ball, and subtracting the centrum of the paddle.

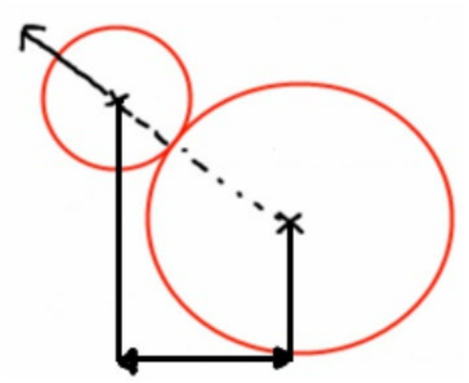

Figure 5: New direction of travel – Calculation for X direction

```
mBallSpeedX = mBallX - mPaddleX;
```

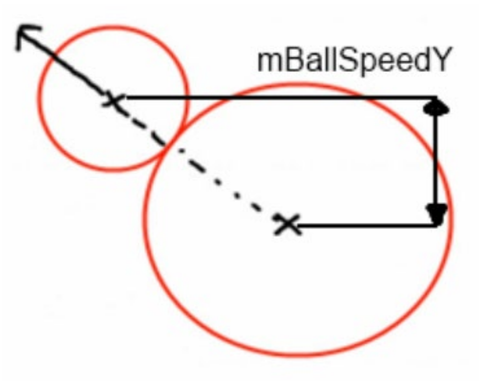

Figure 6: New direction of travel – Y direction

```
mBallSpeedY = mBallY - mCanvasHeight;
```

Thus with the below statements we can get a more accurate direction of speed.

```
mBallSpeedX = mBallX - mPaddleX;

mBallSpeedY = mBallY - mCanvasHeight;
```

Now to calculate speed of the ball before collision:

```
float speedOfBall = (float)Math.sqrt(mBallSpeedX*mBallSpeedX +
mBallSpeedY*mBallSpeedY);
```

Note: Math.sqrt is a Java function that can be used to find the square root. Java has a library of many different and useful mathematical functions. Visit the Oracle website to explore more options.

At this point your added code in **updateGame** will look like:

```
float distanceBetweenBallAndPaddle;

//If the ball moves down on the screen perform potential paddle collision
if(mBallSpeedY > 0) {

      //Get actual distance (without square root - remember?) between the
      mBall distanceBetweenBallAndPaddle = (mPaddleX - mBallX) * (mPaddleX -
      mBallX) +

(mCanvasHeight - mBallY) * (mCanvasHeight - mBallY);


      //Check if the actual distance is lower than the allowed => collision
      if(mMinDistanceBetweenBallAndPaddle >= distanceBetweenBallAndPaddle) {

            //Get the present speed

            float speedOfBall = (float)Math.sqrt(mBallSpeedX*mBallSpeedX +
mBallSpeedY*mBallSpeedY);

            //Change the direction of the ball

            mBallSpeedX = mBallX - mPaddleX;

            mBallSpeedY = mBallY - mCanvasHeight;
            }

      }
```

Let us calculate the new speed:

```
float newSpeedOfBall = (float)Math.sqrt(mBallSpeedX*mBallSpeedX +
mBallSpeedY*mBallSpeedY);
```

IMPORTANT: Make sure to add this line after the statements for change direction of the ball.

Then that can be used to adjust mBallSpeedX and mBallSpeedY so that the speed is exactly the same as the old before the collision.

```
mBallSpeedX = mBallSpeedX * speedOfBall / newSpeedOfBall;

mBallSpeedY = mBallSpeedY * speedOfBall / newSpeedOfBall;
```

Added code in **updateGame** at this point looks like:

```
      float distanceBetweenBallAndPaddle;

      //If the ball moves down on the screen perform potential paddle
collision

       if(mBallSpeedY > 0) {

             //Get actual distance

             distanceBetweenBallAndPaddle = (mPaddleX - mBallX) * (mPaddleX -
            mBallX) + (mCanvasHeight - mBallY) * (mCanvasHeight - mBallY);


      //Check if the actual distance is lower than the allowed => collision
      if (mMinDistanceBetweenBallAndPaddle >= distanceBetweenBallAndPaddle) {

            float speedOfBall = (float)Math.sqrt(mBallSpeedX*mBallSpeedX +

      mBallSpeedY*mBallSpeedY);

             mBallSpeedX = mBallX - mPaddleX;

             mBallSpeedY = mBallY - mCanvasHeight;


      float newSpeedOfBall = (float)Math.sqrt(mBallSpeedX*mBallSpeedX +

mBallSpeedY*mBallSpeedY);


            mBallSpeedX = mBallSpeedX * speedOfBall / newSpeedOfBall;

            mBallSpeedY = mBallSpeedY * speedOfBall / newSpeedOfBall;

      }

 }
```

Apart from bouncing off the paddle, the ball also bounces off the bottom edge of the screen. But we do not want it to be that way. So let us remove the condition that checks the bottom edge of the screen from the **updateGame**.

```
if (mBallY <= mBall.getWidth() / 2 && mBallSpeedY < 0) {

      mBallSpeedY = -mBallSpeedY;

}
```

Now running the game you will see that if you don't play the game the ball will disappear from the bottom edge of the screen. But if this happens that means we have lost the game. Let us add functionality to do this.

The function setState is available to us. We can set the state to indicate that we have lost the game. This needs to go in the updateGame method.

```
if(mBallY >= mCanvasHeight) {

 setState(GameThread.STATE_LOSE);

}
```

Another function **updateScore** is available to increase the score in the game. We can use this to update the score if you hit the paddle.

```
updateScore(1);
```

Above statement should be called in the updateGame method inside the condition where there is a collision happening.

At this point our **updateGame** will look similar to this:

```
protected void updateGame(float secondsElapsed) {

      float distanceBetweenBallAndPaddle;


      //If the ball moves down on the screen perform potential paddle
collision

      if(mBallSpeedY > 0) {

            //Get actual distance (without square root - remember?)

            distanceBetweenBallAndPaddle = (mPaddleX - mBallX) * (mPaddleX -
      mBallX) + (mCanvasHeight - mBallY) * (mCanvasHeight - mBallY);


            //Check if the actual distance is lower than the allowed =>
      collision

            if (mMinDistanceBetweenBallAndPaddle >=
distanceBetweenBallAndPaddle) {
```

```
                float speedOfBall = (float)Math.sqrt(mBallSpeedX*mBallSpeedX
+
mBallSpeedY*mBallSpeedY);
                mBallSpeedX = mBallX - mPaddleX;
                mBallSpeedY = mBallY - mCanvasHeight;


                float newSpeedOfBall =
            (float)Math.sqrt(mBallSpeedX*mBallSpeedX +
mBallSpeedY*mBallSpeedY);


                mBallSpeedX = mBallSpeedX * speedOfBall / newSpeedOfBall;
                mBallSpeedY = mBallSpeedY * speedOfBall / newSpeedOfBall;


                updateScore(1);
            }
        }


        //Move the ball's X and Y using the speed (pixel/sec)
        mBallX = mBallX + secondsElapsed * mBallSpeedX;
        mBallY = mBallY + secondsElapsed * mBallSpeedY;


        if ( ((mBallX <= mBall.getWidth()/2 ) && (mBallSpeedX < 0 )) ||
                ((mBallX >= mCanvasWidth - mBall.getWidth()/2 ) &&
            (mBallSpeedX >
0))){
            mBallSpeedX = -mBallSpeedX;
        }


        //If the ball goes out of the top of the screen and moves towards the
top of the screen
        //change the direction of the ball in the Y direction
        if (mBallY <= mBall.getWidth() / 2 && mBallSpeedY < 0) {
            mBallSpeedY = -mBallSpeedY;
        }
```

```
    mPaddleX = mPaddleX + secondsElapsed * mPaddleSpeedX;
    if(mBallY >= mCanvasHeight) {
              setState(GameThread.STATE_LOSE);
         }
    }
```

Try to add a smiley ball and detect collisions between the ball and the smiley ball. If the smiley ball was hit increase the score (instead of increasing the score for paddle hits).

Answer

If you want to check your answer, open **v4.java** file from the downloaded game framework's **TheGame versions** folder and compare your answer to this.