University of Reading

Guide to conditional statements in the game

Additional Resource

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

If you have not already prepared your project for Week 3 revisit previous step to find instructions on doing it.

This PDF tutorial follows the same sequence as the video and will show you how the changes are done for the X direction (horizontal direction) and once you have seen the video and followed this guide we encourage you to make similar changes to the Y direction (vertical direction).

NOTE: Throughout this tutorial we are referring to **TheGame.java** file

1. Change ball's initial position to (-100, -100).

```
public class TheGame extends GameThread{

    //Will store the image of a ball
    private Bitmap mBall;

    //The X and Y position of the ball on the screen (middle of ball)
    private float mBallX = -100;
    private float mBallY = -100;

    //The speed (pixel/second) of the ball in direction X and Y
    private float mBallSpeedX = 0;
    private float mBallSpeedY = 0;

    //This is run before anything else, so we can prepare things here
    public TheGame(GameView gameView) {
        //House keeping
        super(gameView);
```

Figure 1: Assign – 100 for mBallX and mBallY

2. To make the ball not respond to user inputs, we need to remove code from the methods **actionOnTouch** and **actionWhenPhoneMoved**. This is already done for you on the v3 version of the code.
3. Assign -100 as the ball's speed in X direction in the **setupBeginning** method.

```
public void setupBeginning() {
    //Initialise speeds
    mBallSpeedX = -100;
    mBallSpeedY = 0;

    //Place the ball in the middle of the scre
    //mBall.Width() and mBall.getHeigh() gives
    mBallX = mCanvasWidth / 2;
    mBallY = mCanvasHeight / 2;
}
```

Figure 2: Assign speed for X direction

4. Now let's look at how the ball can leave the screen:

   There are two ways the ball can leave the screen from the X direction: either from the left-hand side (mBallX < 0) or from the right-hand side (mBallX > mCanvasWidth).

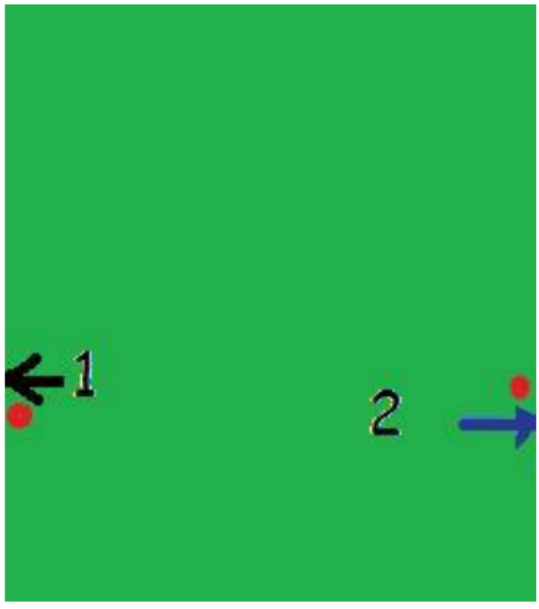

Figure 3: Screen and two possible ways of ball disappearing (X direction)

5. Next we need to fix the problem of the ball leaving the screen. This part of the code should go in **updateGame** method. First we check whether the ball goes out of the screen from left side and if so set the speed to zero.

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

    if (mBallX < 0){
        mBallSpeedX = 0;
    }
}
```

Figure 4: Check ball going out from left side

However, if this condition is used, as we have seen, part of the ball leaves the screen before it changes direction. This is because mBallX refers to the X coordinate of the centre of the ball.

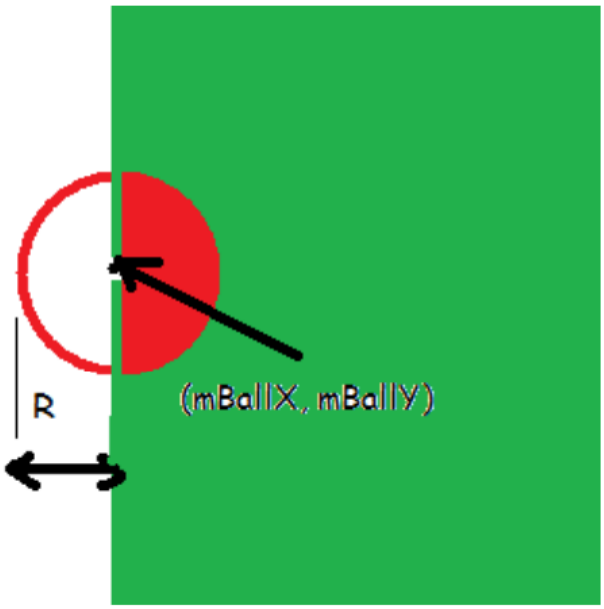

Figure 5: Showing red ball's centre and screen's left side edge

In Figure 5 you can see that if you set the condition as (mBallX < 0), by the time mBallX equals 0 half of the ball has already left the screen. For the ball not to leave the screen you need to make sure you set the condition correctly. So, (mBallX < mBall.getWidth()/2).

Note: getWidth() is a method that gives the width of the image and because you want the radius (R) of the ball you take mBall.getWidth()/2. This getWidth() is a function call, we will cover the topic of functions in week 6.

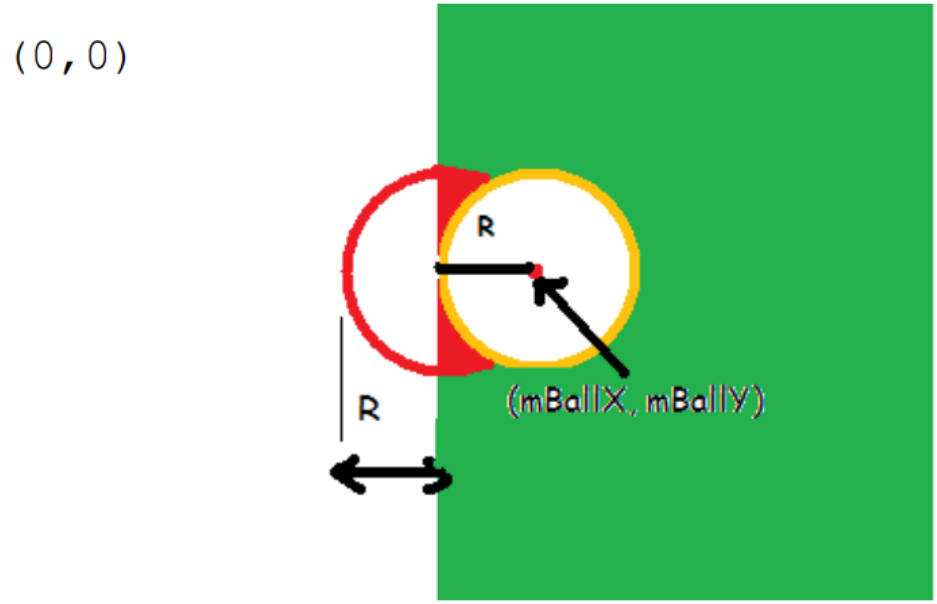

Figure 6: Correcting Condition Diagram

Figure 6 illustrates why mBallX < mBall.getWidth()/2 needs to be used as the condition for the if statement. (R = mBall.getWidth()/2).

Notice that the point where the ball touches the screen on left side the mBallX equals mBall.getWidth()/2.

6. Now refine the condition to avoid the half of the ball going out of the screen.

```
if (mBallX <= mBall.getWidth()/2 ){
      mBallSpeedX = 0;
}
```

7. Next assign a value of 100 as speed instead of assigning zero.

```
if (mBallX <= mBall.getWidth()/2 ){
      mBallSpeedX = 100;
}
```

8. Copy and paste the condition and edit it for right hand edge. Here maximum width is the mCanvasWidth and if the ball is going out of the screen mBallX will be greater than that. Also if with a positive speed the ball leaves the screen, to get it back on to the screen we will have to reverse the direction of travel of the ball.

```
if (mBallX <= mBall.getWidth()/2 ){
      mBallSpeedX = 100;
   }
if (mBallX >= mCanvasWidth - mBall.getWidth()/2 ){
      mBallSpeedX = -100;
}
```

You need to make sure that you change the direction of travel but not the speed of travel. How do we know it is always 100? In order to achieve this you can reverse the current speed by multiplying it by (-1).

(**Note:** (-1) *(100) = -100 and (-1)* (-100) = 100 in fact you can replace 100 by any value.)

9. Reverse the direction of the speed instead of assigning a speed to the ball.

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

10. Allocate a third of mCanvasWidth as the speed at the beginning instead of 100.

```
mBallSpeedX = -mCanvasWidth/3;
```

Look at the two if statements they look very similar. One condition checks for the left side of the screen, the other checks for the right side of the screen. In both cases we invert the speed. Now let's look at how a combined if statement can be created that would do both.

Remember learning about the logical AND operator, and the logical OR operator in week 2? This is where you are going to use them.

Here the logical OR operator is used, because when either condition is true the speed of mBall needs to be inverted:

11. Let us put the two logical statements together using OR operator (||). Be careful to close all the open parenthesis or the system will complain of errors.

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

12. Enhance the logic to consider the travel direction

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

13. Next we need to do a similar process to check the ball goes out of the screen on the Y direction. First allocate a speed of mCanvasHeight/3 to mBallSpeedY variable in the **setupBeginning** method.

```
      mBallSpeedY = mCanvasHeight/3;
```

14. Copy and paste the logic for X direction and try to change them for Y direction. Don't forget that where in X direction we look at Width for Y direction we need to look at Height.

(**Hint:** instead of mBallX you will be working with mBallY; instead of mBallSpeedX it will be mBallSpeedY; instead of mCanvasWidth it will be mCanvasHeight).

Answers:

In **updateGame** to check whether the ball goes out of the top of the screen.

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

Similarly for bottom of the screen.

```
if(mBallY >= mCanvasHeight - mBall.getWidth() / 2 &&
mBallSpeedY > 0) {
     mBallSpeedY = -mBallSpeedY;
}
```

You can use OR to put both conditions together as we have seen in the case of the X direction.

```
if ((mBallY <= mBall.getWidth() / 2 && mBallSpeedY < 0)||
(mBallY >= mCanvasHeight - mBall.getWidth() / 2 &&
mBallSpeedY > 0) ) {
     mBallSpeedY = -mBallSpeedY;
}
```