Disclaimer: The author takes no responsibility for any conclusions you may draw from using this demo.

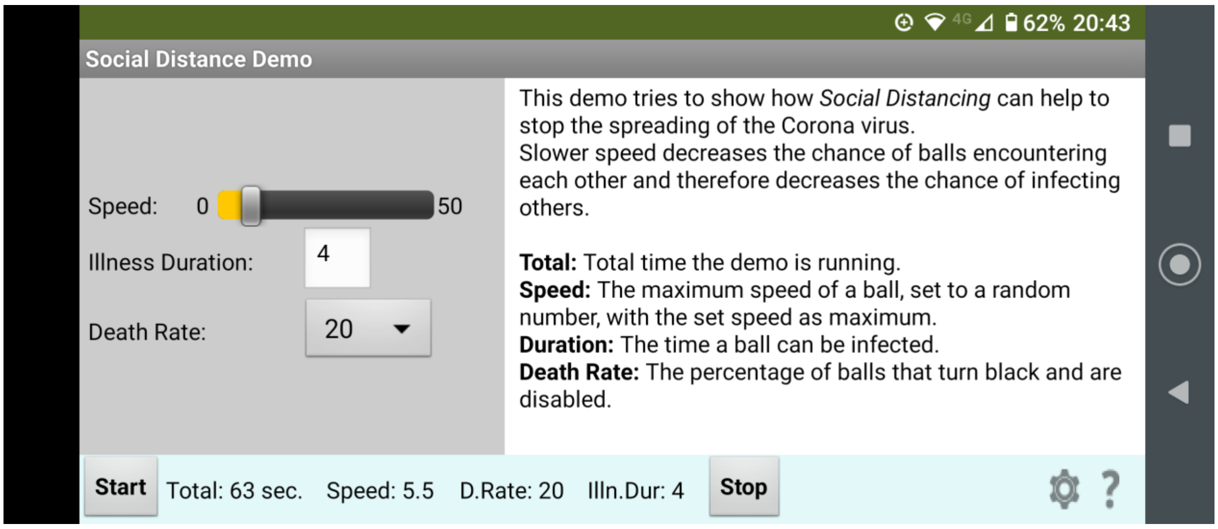
**Basically, this is an example of ball animation on a canvas, the use of any-blocks and any-events.**

Author: Ghica van Emde Boas

# Social Distancing Demo – part 1

This document describes a demo that can demonstrate the effect of social distancing on the speed of spreading a virus. This is done by allowing a set of balls to move freely and randomly across the phone screen. If they encounter another ball, one ball may infect another ball and so on.

Balls may recover after a set period and they can die. In the las case the ball will turn black and become disabled.


62% 20:43

Social Distance Demo

Speed: 0 50

Illness Duration: 4

Death Rate: 20

This demo tries to show how *Social Distancing* can help to stop the spreading of the Corona virus.
Slower speed decreases the chance of balls encountering each other and therefore decreases the chance of infecting others.

**Total:** Total time the demo is running.
**Speed:** The maximum speed of a ball, set to a random number, with the set speed as maximum.
**Duration:** The time a ball can be infected.
**Death Rate:** The percentage of balls that turn black and are disabled.

Start Total: 63 sec. Speed: 5.5 D.Rate: 20 Illn.Dur: 4 Stop

In the full demo you can set the maximum speed of a ball, the duration that a ball can be ill. After that, the ball can die according to the death rate set. The demo ends when there are no infected balls left. It is clear that at higher speeds the demo will end sooner, but more balls will become infected and may die.

Here in Part 1, we look first at a *simplified version*, were balls can become infected but they never heal. Eventually all balls will become infected and the demo ends. You can see that the infection rate may start slow, but as soon as more balls are infected their number will grow rapidly. At the end it may take a while until the last ball is caught.

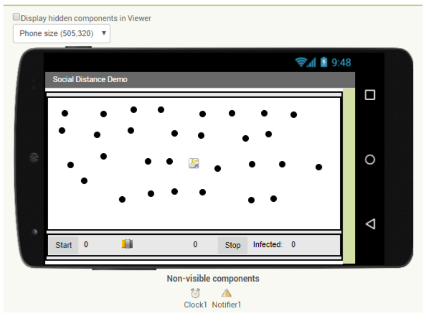
This is like the scenario many countries have: by making everybody move slowly or not at all, the time it takes until everybody is infected will increase, allowing healthcare to keep up with the demand.

This is a description of the app. I just explain what the blocks do, assuming that you already know what the components do.

The finished app that will be described in Part 2, can be found here:
[ai2.appinventor.mit.edu/?galleryId=6696333792116736](ai2.appinventor.mit.edu/?galleryId=6696333792116736)

## Screen Design

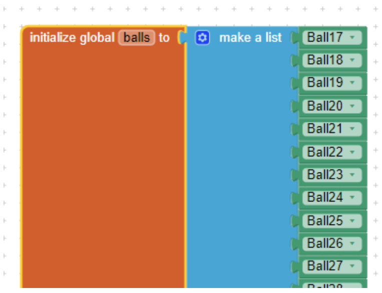
See the screen design below:



If you want to follow along and skip making the design, you can use this aia: socialDistance_01a_nb.aia

The finished app is here: socialDistance_01a.aia

As you can see in the design, there is a Canvas component with 30 balls on it. The balls are defined like this:

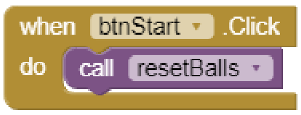


```
initialize global balls to make a list  Ball17
                                        Ball18
                                        Ball19
                                        Ball20
                                        Ball21
                                        Ball22
                                        Ball23
                                        Ball24
                                        Ball25
                                        Ball26
                                        Ball27
```

Don’t worry that the balls are not sorted (I improved that in a later version), but notice that the ball components are immediately added during initialization of the *balls* variable. App Inventor veterans may think that this is not possible, but since about half a year ago it is. Very practical!

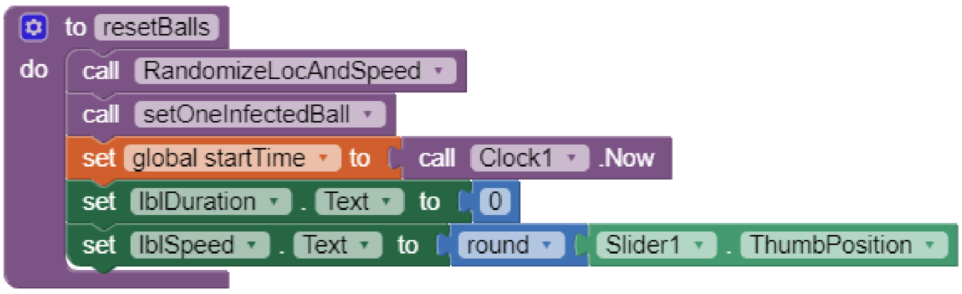
## Starting the Demo

There is no *Screen.Initialize* block needed. The way to start the demo is to click the Start button:



```
when btnStart .Click
do  call resetBalls
```

The procedure *resetBalls* will be called:



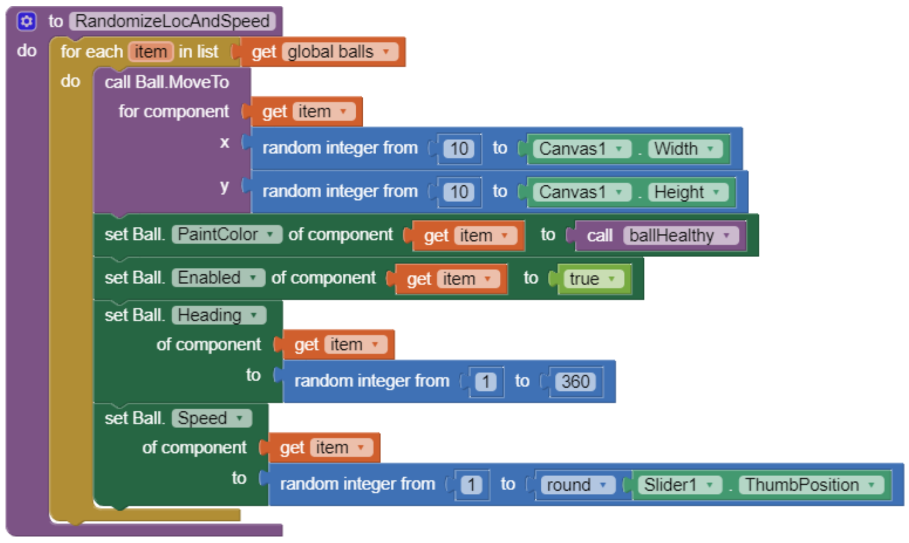
```
to resetBalls
do  call RandomizeLocAndSpeed
    call setOneInfectedBall
    set global startTime to call Clock1 .Now
    set lblDuration . Text to 0
    set lblSpeed . Text to round Slider1 . ThumbPosition
```

This procedure will distribute all balls randomly over the Canvas and give them a random direction and speed. The maximum speed is given by the position of the thumb on the slider. Healthy balls are blue.

When that is done, a ball, chosen randomly, is infected and therefore it will turn red. The duration indication is reset.

## Randomize Location and Speed

The randomize location and speed procedure will iterate through the list of balls and for each:

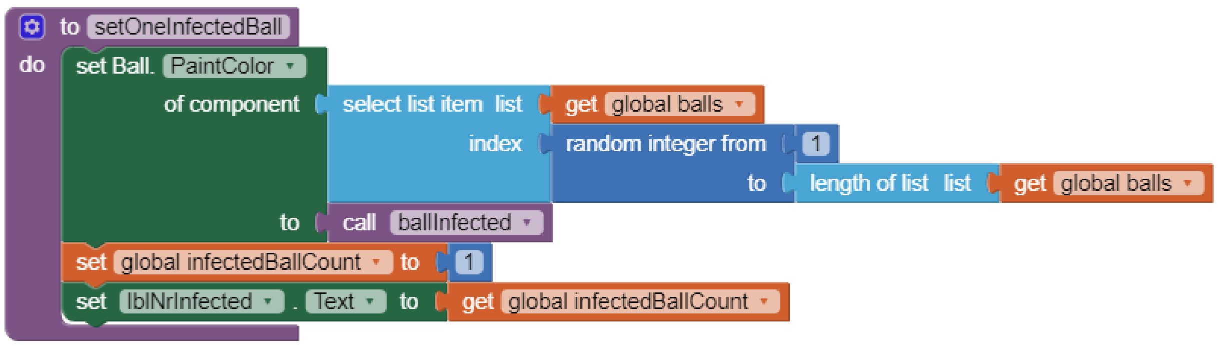


```
to RandomizeLocAndSpeed
do  for each item in list  get global balls
    do  call Ball.MoveTo
            for component  get item
                        x  random integer from 10 to Canvas1 . Width
                        y  random integer from 10 to Canvas1 . Height
        set Ball. PaintColor of component  get item  to  call ballHealthy
        set Ball. Enabled of component  get item  to  true
        set Ball. Heading
            of component  get item
                      to  random integer from 1 to 360
        set Ball. Speed
            of component  get item
                      to  random integer from 1 to round  Slider1 . ThumbPosition
```

- The ball will be positioned at a random location on the canvas
- The ball will get a healthy (blue) color.
- The ball will be enabled. So far that will always be the case, but later, when balls can die, this is needed to reset them.
- Let the ball move in a random direction
- Set the speed of the ball randomly, with a maximum determined by the slider thumb position.

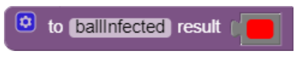
## Set one Infected Ball

Randomly a ball is selected from the list of balls and painted red.



```
to setOneInfectedBall
do  set Ball. PaintColor
        of component  select list item  list   get global balls
                                        index  random integer from 1
                                                                to  length of list  list  get global balls
                  to  call ballInfected
    set global infectedBallCount to 1
    set lblNrInfected . Text to  get global infectedBallCount
```

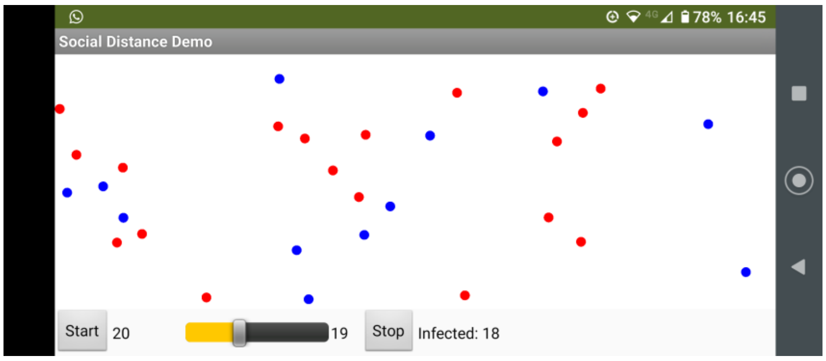
We should note, that we used abstractions for the colors. This means that if we would change our mind about infected balls being red, we could just change the color in the procedure *ballInfected*.



```
to ballInfected result 🟥
```

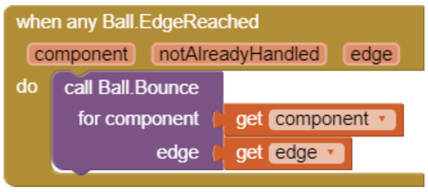
## The Running App

After a few seconds the phone screen may look like this:



## Edge Reached

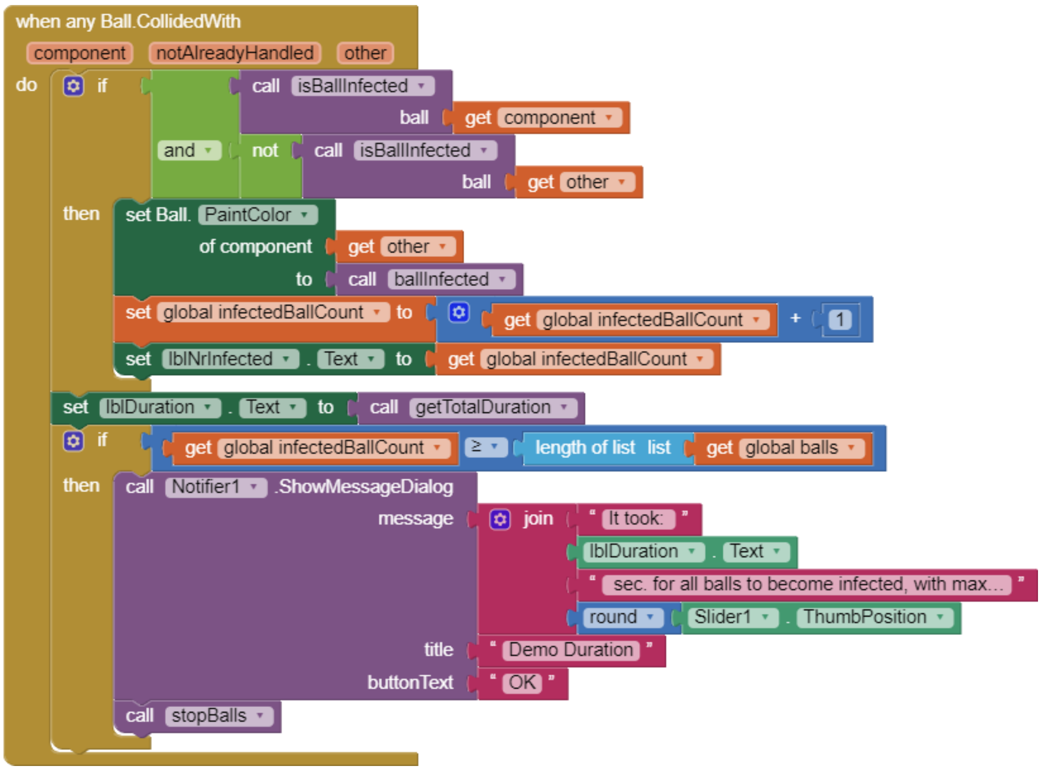
Balls will hit the border and in that case they will bounce:



```
when any Ball.EdgeReached
  component  notAlreadyHandled  edge
do  call Ball.Bounce
      for component  get component
      edge  get edge
```

## Ball Collision

Each time a ball encounters another ball, it will infect the other ball if it was infected itself and the other was not.

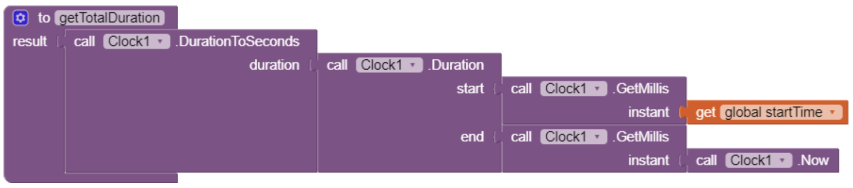


```
when any Ball.CollidedWith
  component  notAlreadyHandled  other
do  if    call isBallInfected
              ball  get component
          and
          not  call isBallInfected
                  ball  get other
    then  set Ball. PaintColor
              of component  get other
              to  call ballInfected
          set global infectedBallCount to  get global infectedBallCount + 1
          set lblNrInfected . Text to  get global infectedBallCount
    set lblDuration . Text to  call getTotalDuration
    if    get global infectedBallCount ≥ length of list list  get global balls
    then  call Notifier1 .ShowMessageDialog
              message  join  " It took: "
                             lblDuration . Text
                             " sec. for all balls to become infected, with max... "
                             round  Slider1 . ThumbPosition
              title  " Demo Duration "
              buttonText  " OK "
          call stopBalls
```

As you can see, a count will be kept of how many balls are infected. If that count is the same as the number of balls in the list, the demo should stop and a message is shown with the duration of the demo and the maximum speed of the balls.

## Duration

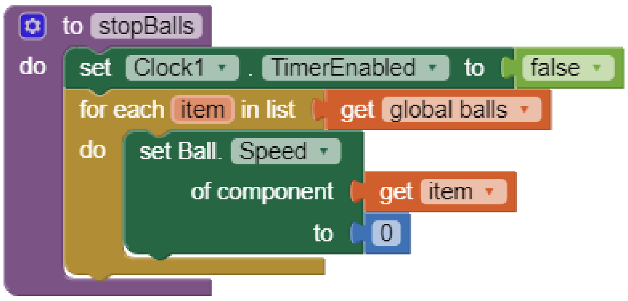
The duration the demo has taken is determined at each collision and show at the bottom of the phone screen.



```
to getTotalDuration
result  call Clock1 .DurationToSeconds
                duration  call Clock1 .Duration
                                  start  call Clock1 .GetMillis
                                                   instant  get global startTime
                                  end    call Clock1 .GetMillis
                                                   instant  call Clock1 .Now
```

## Stopping Ball Movement

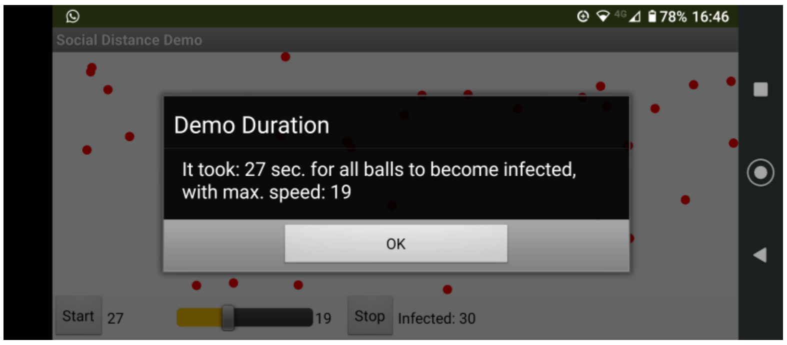
The balls can be stopped by clicking the Stop button, They will be stopped when all balls are infected, at the end of the demo.



```
to stopBalls
do  set Clock1 . TimerEnabled to  false
    for each item in list  get global balls
    do  set Ball. Speed
                of component  get item
                          to  0
```

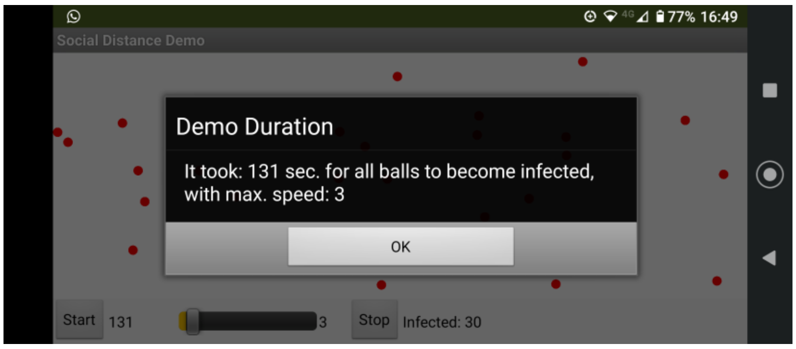
## Demo Results

Here are two typical results, one for a moderate speed of the balls, one for a slow speed.



Demo Duration

It took: 27 sec. for all balls to become infected,
with max. speed: 19



Demo Duration

It took: 131 sec. for all balls to become infected,
with max. speed: 3