Graphics and Animation (Tkinter)

A Starry Night

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Intro

- To build on what we know about the tkinter library
- To discuss the Canvas class
  - Creating and Updating
- To discuss the objects available for the Canvas Class
  - Creating and Updating
- To discuss the coordinate systems
  - Getting coordinates of an object
  - Updating the coordinates of an object
- Concluding with collision detection
  - And what to do when objects collide

Coursework 2

- To create a game using tkinter
- The game must incorporate:
  - images
  - shapes
  - animations
  - a scoring mechanism
  - a leaderboard
  - collision detection

The Tk Class

- The Tk class is used to create a top level window
- It is this object that interfaces with Tk
- If you create multiple windows, each window will have it’s own interpreter to Tk
- There are many types of widgets that can be used with TkInter
  - Button, Canvas, Checkbutton, Entry ...
- Today we will focus on the Canvas widget

The Canvas Widget

- The Canvas Class allows us to create shapes and text to be placed on the canvas.
- When creating a shape it is bound to a box (a rectangle)
- Two types of coordinate systems are used, the window coordinate system with (0,0) being top left and the canvas coordinate system which specify where items are drawn

```
canvas = Canvas(window,width=400,
height=400, bg="black")
```

The General Structure

- We import the things that we want to use from tkinter
- We create a new instance of the Tk() method
- When we create a canvas, we don't have to specify any parameters when creating the canvas
- When working with a canvas we generally use the pack geometry manager

The Canvas

- Once we have created a canvas we can configure various options
  - Border (yes, no and size), also Type (sunken, Raised, Groove and Ridge)
  - Background Colour
  - Scroll Region defined how large an area of the canvas can be scrolled.
  - Confined Scroll Region (true by default meaning the canvas cannot be scrolled outside a scroll region)
  - Cursor (we can change the mouse cursor to another shape, for example, a pirate skull and cross bones)
  - The height and width
  - A highlight colour (the colour shown in the focus highlight)
  - … There are more

Creating Objects for the Canvas

```
oval = canvas.create_oval(x0, y0, x1, y1, options)
```

- We must provide the bounding box (rectangle) coordinates
- Options are optional and can be included when creating the object

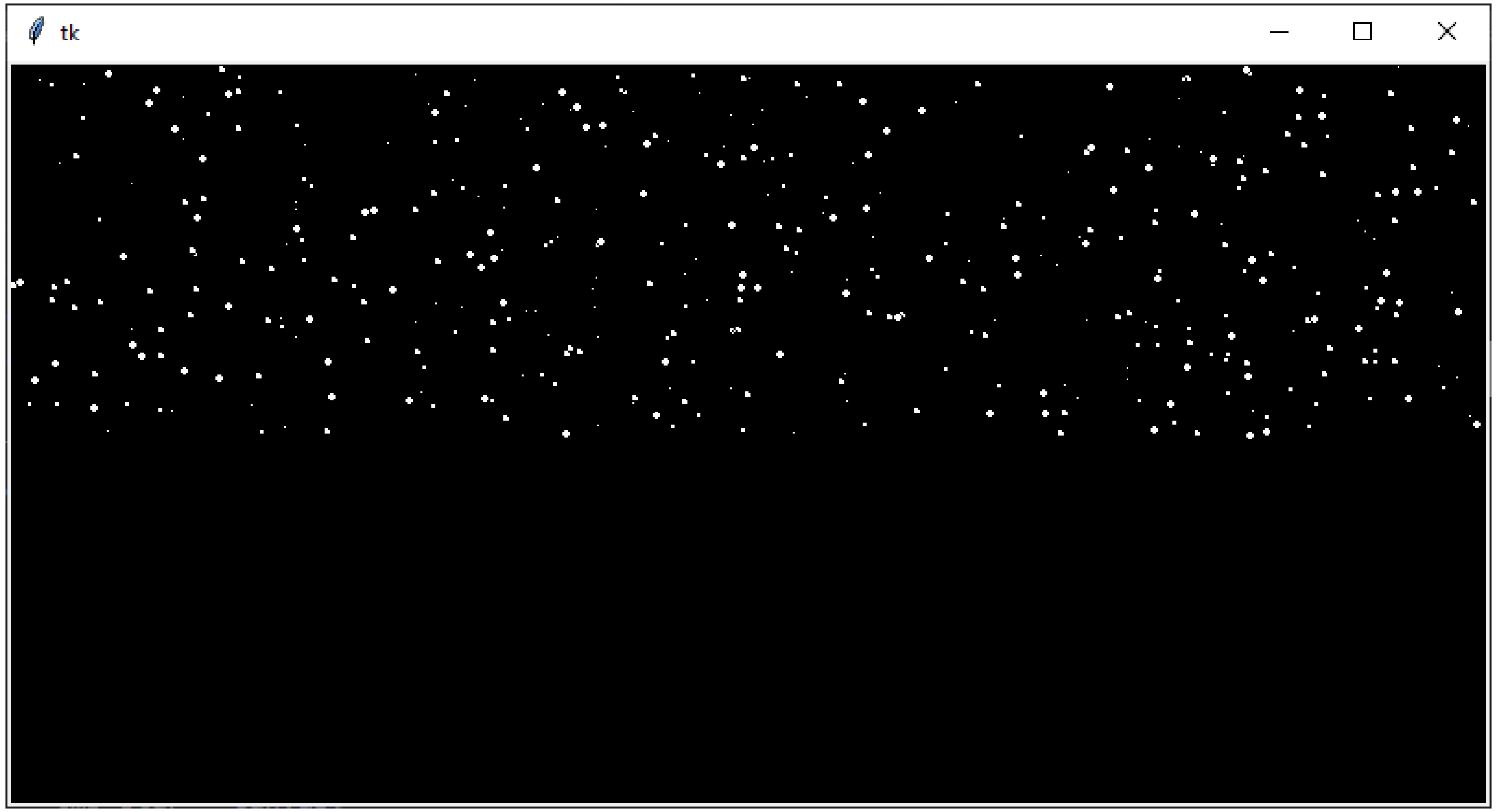
```
xy = (0,0,50,50)
circle = canvas.create_oval(xy,
                            fill="red")
```

canvas.config vs canvas.itemconfig

- The configuration of the canvas can be changed using the canvas method named config
- Items created on the canvas can be reconfigured using itemconfig method
- Both methods belong to the canvas class
- To delete an item from the canvas you can use the canvas delete method

Let’s try to put it together

- I want to create a starry night
- Stars are of different size and different brightness
- I want 400 stars in the sky



```
star.append(tmp_star)
```

The Pseudocode

```
Import the required modules
Create a window and canvas of desired dimensions
Display them
Configure background colour of the canvas to black
Create a list to store the stars
Create a list of colours named c
star_count ← 0
while (star_count < 400)
   begin
      get random x, y positions
      get a random size and colour
      create an oval at position x,y of desired size
      give it a fill colour
      star_count ← star_count + 1
   end
```

So far we have seen…

- How to create a root object (window) and a canvas
- How to use **config** to configure the canvas object
- How to create an oval using a method from the canvas class
- How to use **itemconfig** to reconfigure an object belonging to the canvas class

Graphics and Animation (Tkinter)

A Starry Night

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Graphics and Animation (Tkinter)

Coordinates

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Intro

- How to bind a mouse key with a motion event that will call a function to deal with that event
- How to reconfigure an item so that when it is active the fill colour will change
- How to create some text for the canvas
- How to access the current x,y postion of the mouse pointer
- How to get the coordinates of an object that uses both windows and canvas coordinate systems
- How to update the position of an object

The coordinates method

- The coords method serves two purposes
  - Get the current coordinates of an object
  - Update the current coordinates of an object
- We can change the position of the canvas objects, but not directly
  - Objects created on the canvas class don’t have any methods
  - But they do have options (the options are accessed and controlled by the canvas class)
- So… if we had an oval object named star, we can’t do this:

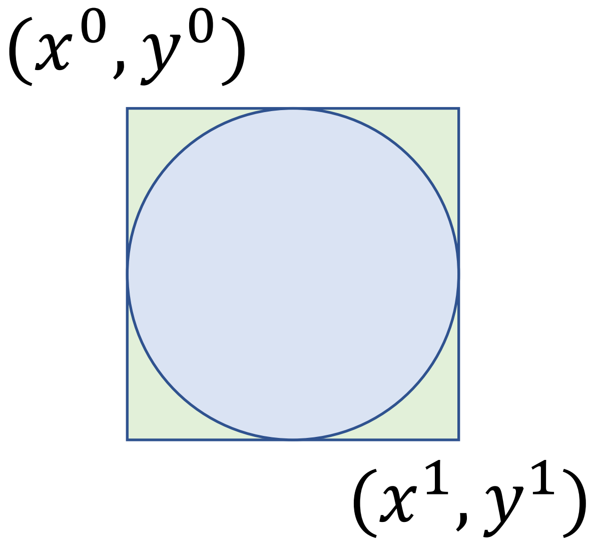
```
position = star.coords()
```

- But we can do this:

```
position = canvas.coords(star)
```

Coordinates

- Most shapes have four associated coordinates (arc, bitmap, image, oval, rectangle and window)
- More complex shapes like polygons must have at least three vertices
- Other object might use the windows coordinate system (i.e. text)
- The coordinates are stored in a list

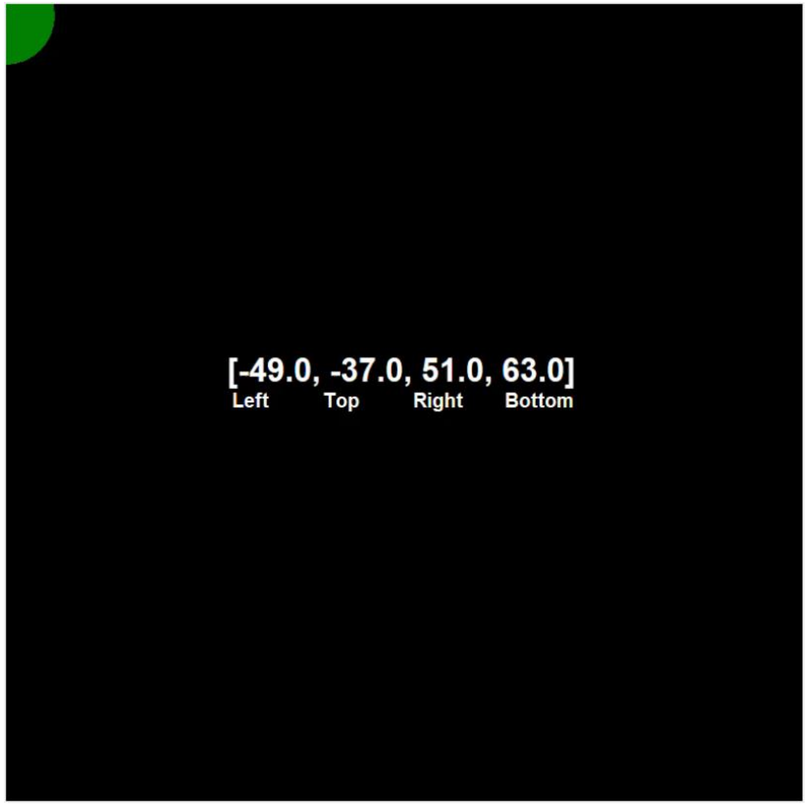


$$[x^0, y^0, x^1, y^1]$$

Let’s try and put it together

- I want to create a circle that follows the mouse pointer. As it follows the mouse pointer I want to display the current coordinates in the center of canvas for the circles position

- This allows us to:
  - Access coordinates of an object
  - Update coordinates of an object
  - Create a motion bind to the canvas
  - Create some text on the canvas



```
[-49.0, -37.0, 51.0, 63.0]
 Left    Top   Right Bottom
```

The Pseudocode

```
coords_label = canvas.create_text(
    400,400,
    text="Left      Top      Right        Bottom",
    fill="white", font=("Arial Bold",15))
```

```
Create a motion bind with the left mouse button and call the mouse_move function when LMB clicked
Define the coordinates for the circle
Create a green circle
Reconfigure the circle so it changes to red when active
Create the coordinate text
Create the coordinate labels
```

```
canvas.coords(coords_text, xOffset,350)
```

The Pseudocode

```
function mouse_move(event)
   begin
      get x and y of the mouse pointer from the event
      update the coordinates of the circle
      get the new coordinates of the circle
      update the coordinate text with the new coordinates
      get the coordinates of the coordinate text
      calculate a new center position for the coordinate text
      update the position of the coordinate text
   end
```

So now we know…

- How to bind a mouse key with a motion event that will call a function to deal with that event
- How to reconfigure an item so that when it is active the fill colour will change
- How to create some text for the canvas
- How to access the current x,y postion of the mouse pointer
- How to get the coordinates of an object that uses both windows and canvas coordinate systems
- How to update the position of an object

Graphics and Animation (Tkinter)

Coordinates

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter)

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Intro

- Now that we know how to create an object
- Access the coordinates of the object
- Update the coordinates of the object
- We can create some simple animations
- We essentially change the x or y coordinates (or both depending on the objective).

Moving an Object

$[x^0, y^0, x^1, y^1]$

- If we had an object called ball we could move the ball to the right by incrementing both the first and third elements of the coords list
- To move the ball to the left, we decrement both the first and third elements
- To move the ball down, we increment both the second and fourth elements
- To move the ball up, we decrement both the second and fourth elements
- To move in a diagonal, we increment all elements ↘
- Or , decrement all elements ↖
- Or, increment x element and decrement y elements ↗
- Or, decrement x elements and increment y elements ↙

Moving by x, y

- Moving on the x,y plane is quite common, therefore, canvas has a method called move which simplifies the process

```
canvas.move(object, x, y)
```

x = 10, y = 0 $\rightarrow$

x = 0, y = 10 $\downarrow$

x = 10, y = 10 $\searrow$

x = 10, y = -10 $\nearrow$

x = -10, y = 0 $\leftarrow$

x = 0, y = -10 $\uparrow$

x = -10, y = -10 $\nwarrow$

x = -10, y = 10 $\swarrow$

Let’s look at a simple implementation

- We want to place a ball on the canvas near the top
- We want to animate the ball falling
- When the ball reaches the bottom we want it to stop

- This allows us to:
  - Create an abject
  - Update the objects coordinates
  - Look at how we can control the speed using sleep
  - Look at how we can update the main window

```
window.update()
```

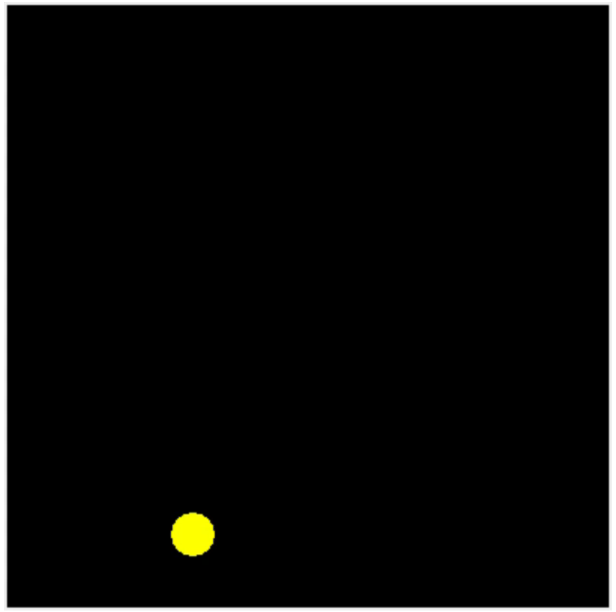
The Pseudocode

```
create a ball
display the ball
while (True)
   begin
      pos ← coords(ball)
      if (pos[3] > 400) then
         begin
            break out of loop
         end
      move(ball, 0, 1)
      wait a short time
      update the window
   end
```

$$[x^0, y^0, x^1, y^1]$$

Simple Collision Detection

- Instead of breaking out of the loop we will allow the ball to bounce when it hits the bottom of the window and the top of the window.
- We just need to update y to either 1 or -1



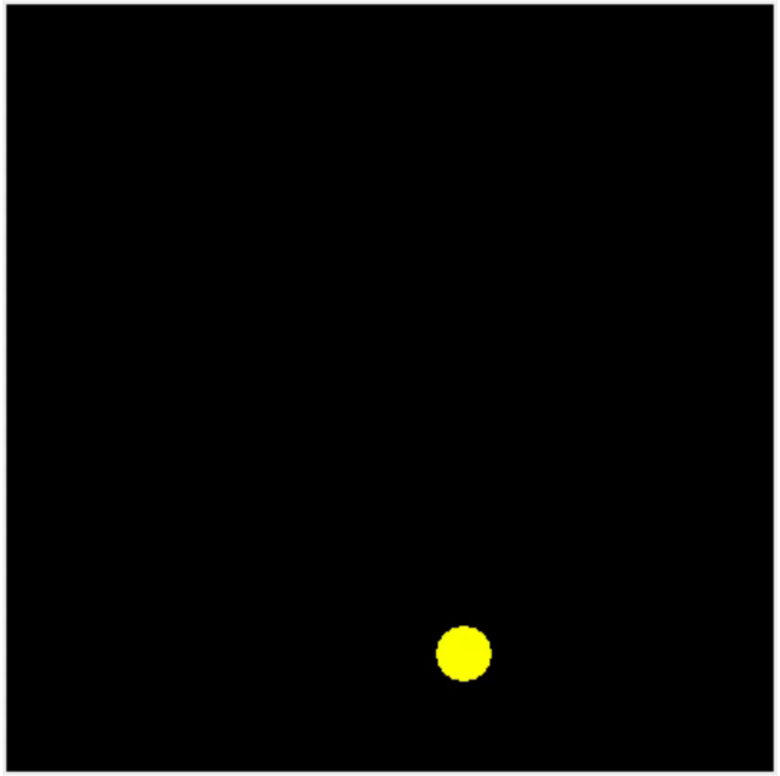
The Pseudocode

$$[x^0, y^0, x^1, y^1]$$

```
create a ball
display the ball
y ← 1
while (True)
   begin
      pos ← coords(ball)
      if (pos[3] > 400  or pos[1] < 0) then
         begin
            y ← -y
         end
      move(ball, 0, y)
      wait a short time
      update the window
   end
```

All four walls

- Hopefully it should be clear that we can bounce off all four walls
- We just need to update incorporate our x coordinate



The Pseudocode

$$[x^0, y^0, x^1, y^1]$$

```
create a ball
display the ball
y ← 1

while (True)
   begin
      pos ← coords(ball)
      if (pos[3] > 400  or pos[1] < 0) then
         begin
            y ← -y
         end

     move(ball, x, y)
      wait a short time
      update the window
   end
```

```
x ← 1
```

```
if (pos[0] < 0  or pos[2] > 400) then
   begin
       x ← -x
   end
```

Conclusion

Now that we know how:

- to create an object
- access the coordinates of the object
- update the coordinates of the object
- create some simple animations by changing the x or y coordinates (or both depending on the objective).

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter)

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter) with a bit of user input

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

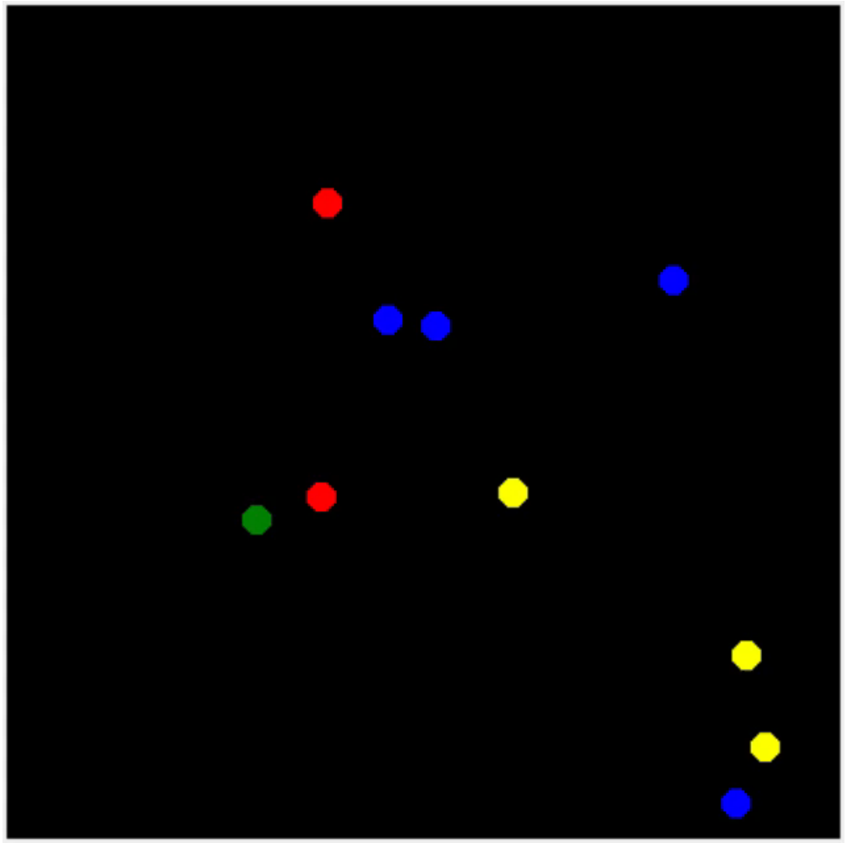
Intro

- We have created the appropriate number of balls
- We placed each ball in a random location
- Selected a random colour
- And set the size of the ball to the diameter specified from the user

Let’s add some complexity

- The user can specify the number of balls and the size
- The balls start in random positions
- Balls are assigned a random colour

- This would be much easier if we implemented a class, but for now we will stick with lists
  - Object Orient programming is coming soon!



How to solve?

- We solve any complex problem in stages
- Fortunately we have the solution already (for one ball)
- We just need a mechanism for the other balls
- Lets start by dealing with the balls

A little change to the Pseudocode

```
create a ball
```

Maps to:

```
ball = canvas.create_oval(110,10,140,40,fill="yellow")
```

So I will refine my pseudocode to match a little better.

```
create_ball(xy,fill="yellow")
```

The Pseudocode (lots of balls)

```
num_balls ← input("Number of Balls: ")
ball ← []
while (num_balls > 0)
   begin
      xy = (110,10,140,40)
      ball.append(create_ball
                  (xy,fill="yellow"))
      num_balls ← num_balls - 1
   end
```

The Pseudocode (random colour)

Next we deal with choosing a random colour.

```
num_balls ← input("Number of Balls: ")
ball ← []
while (num_balls > 0)
    begin
        xy = (110,10,140,40)
        ball.append(create_ball(xy,fill="yellow"))
        num_balls ← num_balls - 1
    end
```

The Pseudocode (random colour)

```
num_balls ← input(“Number of Balls: “)
ball ← []
colour ← ["red", "yellow", "green", "blue"]
while (num_balls > 0)
    begin
        c_col ← rand(0,3)
        xy = (110,10,140,40)
        ball.append(create_ball
                    (xy,fill=colour[c_col]))
        num_balls ← num_balls - 1
    end
```

Create a list of colours

Generate a random number. Store in variable chosen colour (c_col)

Get the appropriate chosen colour from the colour list

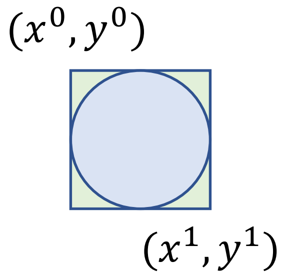
The Pseudocode (start position)

We do something similar to get the start position

```
num_balls ← input("Number of Balls: ")
ball ← []
colour ← ["red", "yellow", "green", "blue"]
while (num_balls > 0)
    begin
        c_col ← rand(0,3)
        xy = (110,10,140,40)
        ball.append(create_ball
                        (xy,fill=colour[c_col]))
        num_balls ← num_balls - 1
    end
```

The Pseudocode (random position)

```
num_balls ← input("Number of Balls: ")
ball ← []
colour ← ["red", "yellow", "green", "blue"]
while (num_balls > 0)
   begin
      c_col ← rand(0,3)
      x = rand(0, 400)
      y = rand(0, 400)
      xy = (x,y,x+25,y+25)
      ball.append(create_ball(xy,fill=colour[c_col]))
      num_balls ← num_balls - 1
   end
```



$$[x^0,y^0,x^1,y^1]$$

Random x start position, random y start postion.

Plug the values into the xy coordinate variable.

+25 is the diameter of the ball

The Pseudocode (random position)

All that remains for the balls is to get the diameter from the user.

```
num_balls ← input("Number of Balls: ")
ball ← []
colour ← ["red", "yellow", "green", "blue"]
while (num_balls > 0)
   begin
      c_col ← rand(0,3)
      x = rand(0, 400)
      y = rand(0, 400)
      xy = (x,y,x+25,y+25)
      ball.append(create_ball
                 (xy,fill=colour[c_col]))
      num_balls ← num_balls - 1
   end
```

The Pseudocode (diameter)

```
num_balls ← input("Number of Balls: ")
diameter ← input("Diameter: ")
ball ← []
colour ← ["red", "yellow", "green", "blue"]
while (num_balls > 0)
   begin
      c_col ← rand(0,3)
      x = rand(0, 400)
      y = rand(0, 400)
      xy = (x,y,x+diameter,y+diameter)
      ball.append(create_ball(xy,fill=colour[c_col]))
      num_balls ← num_balls - 1
   end
```

So far…

- We have created the appropriate number of balls
- We placed each ball in a random location
- Selected a random colour
- And set the size of the ball to the diameter specified from the user

- Next we deal with moving each ball

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter) with a bit of user input

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter) with a bit of user input

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Intro

- We will move each ball
- Bounce if hits edge of screen
- We will see a couple of ifs in a for and the for is inside a while true.

- So the code is a little complex, but hopefully since it is fairly short you will be able to follow along and understand what is happening.

The Pseudocode

```
ball ← create num_balls balls
display the ball
y ← [1] * num_balls
x ← [1] * num_balls
while (True)
   begin
      for i in range(num_balls)
         begin
            pos ← coords(ball[i])
            if (pos[3] > 400  or pos[1] < 0) then
               begin
                  y[i] ← -y[i]
               end
            if (pos[0] < 0  or pos[2] > 400) then
               begin
                  x[i] ← -x[i]
               end
            move(ball[i], x[i], y[i])
         end
   wait a short time
   update the window
end
```

Conclusion

- moved each ball
- Bounces if hits edge of screen
- We will see a couple of ifs in a for and the for is inside a while true.

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter) with a bit of user input

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter) and very simple collision detection

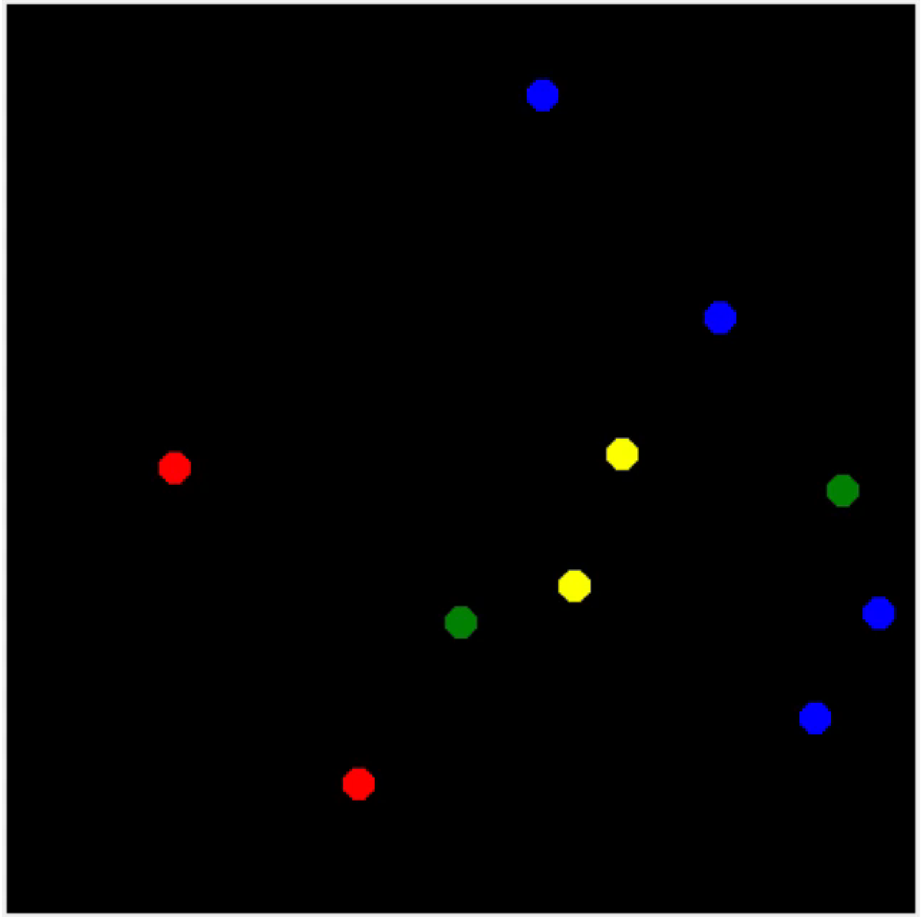
Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1

Intro

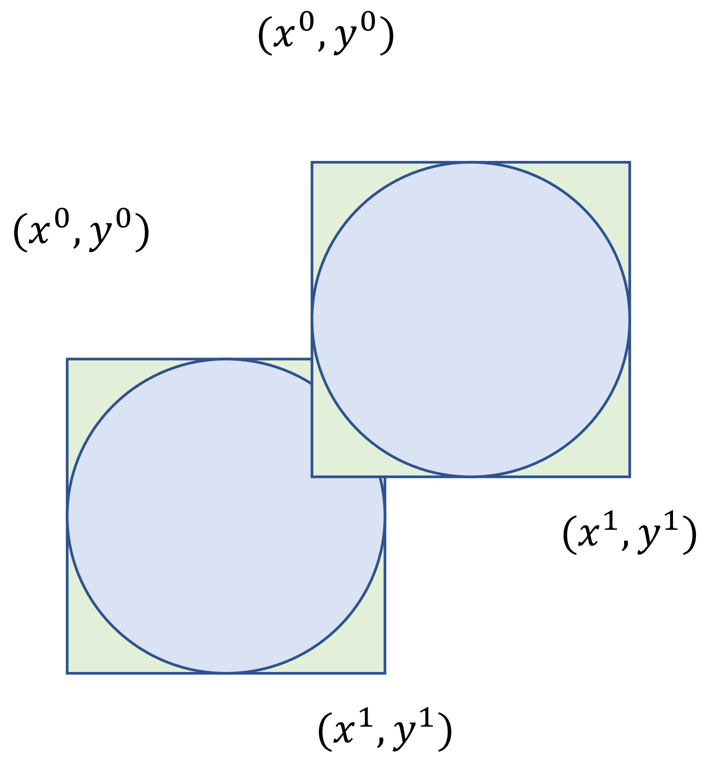
Collision of Balls

- Currently the balls pass through each other
- This is not right
- Lets see how we can implement collision detection between these objects
- This is not the most elegant solution, but it works to some extent



The Pseudocode

```
for j in range(num_balls)
   begin
      if (j == i) then
         begin
            move to next ball
         end
      pos2 ← coords(ball[j])
      if (pos[0] < pos2[2] and pos[2] > pos2[0] and pos[1] <
          pos2[3] and pos[3] > pos2[1]) then
         begin
            y[i] = -y[i]
            x[i] = -x[i]
            y[j] = -y[j]
            x[j] = -x[j]
         end
   end
```



To Conclude

- There was some tricky problems to solve
- We have covered quite a lot
- Don’t worry if you didn’t fully understand everything

- Take homes:
  - We create root object
  - We create a canvas object for the root object
  - We can create objects for the canvas
  - We can access update the objects options
  - We can access the coordinates of an object on the canvas (and update it)
  - We can use lists to store many of the same type of object
  - We can check if the object position is at a boundary (and send it the other way)
  - We can check if an object overlaps another object

Graphics and Animation (Tkinter)

Graphics and Animation (Tkinter) and very simple collision detection

Dr Stewart Blakeway
Lecturer in Computer Science

COMP16321
Introduction to Programming 1