23

Network Games

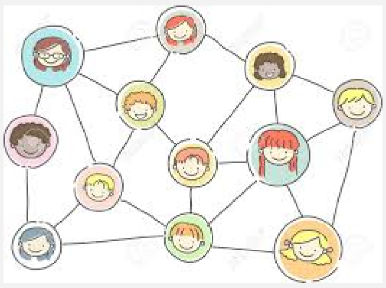
Introduction

There are lots and lots of games that are based around the idea of a network. In this worksheet we (or you) are going to build a game around the idea of trying to click the nodes in a network — in correct order — within a time limit.

As in our fruit ninja and collector games we will develop multiple versions as we add more and more features. The features for this game are similar to what you have implemented for the earlier games so if you get stuck, check back to what you did before or, of course, ask one of us.

What is a 'network'?

A **network** is a collection of things, called **nodes**, with connections, called **arcs**, between various pairs of nodes.



For example, the above is a representation of a network of 12 people with arcs between pairs of people who are friends.

① **Basic network game** .... `network_basic.py`

Our starting version of the game consists of 10 dots (or nodes using network terminology) appearing at random positions on the screen and the player has to click on the dots in correct order. As the player clicks on the dots, lines (or arcs in network terminology) are created to show progress. If the player misses or clicks on a wrong dot the path created is removed and the player has to restart.

The steps given here come from the excellent *Coding Games with Python* book.

② **Adding sounds** .... `network_sound.py`

Go to www.zapsplat.com, and find some sounds that we can use
when you click on the correct dot and when you click on an incorrect dot, and when you finish the sequence. You could do this at home and in the coderdojo session we will show you how to convert music formats to suit pgzero.

For background music have a look at www.melodyloops.com
This has a good selection of tracks and can cut a track to whatever length you want — you could set the length to match the time given to complete the level.

③ **Refactor game** .... `network_refactored.py`

Again, before we move on it is good to see how we can impove our code.

④ **More advanced games**

Here we look at three ways we can build a full game from our starting network game idea.

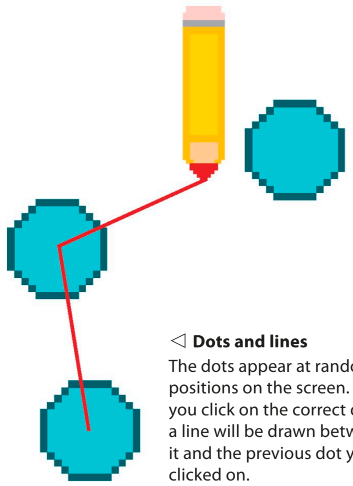
① network_basic

How to build Follow the Numbers

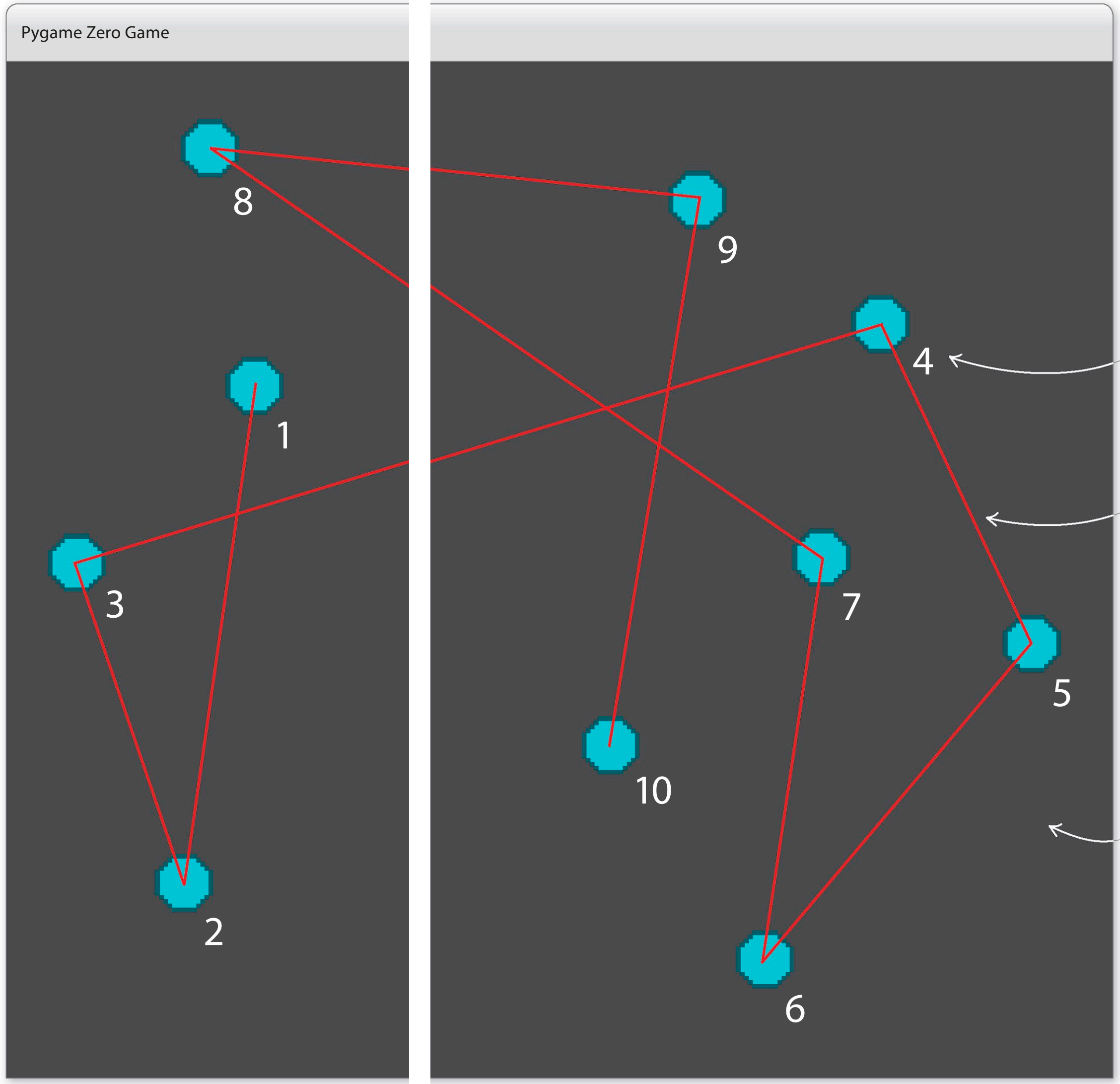
Can you connect all the dots in the correct order? Challenge yourself to finish the game as quickly as you can. Be careful, however—one wrong click and you'll have to start all over again.

What happens

At the beginning of the game, ten dots appear at random positions on the screen, each with a number next to it. You need to click on the dots in the correct order to connect them. The game will finish once you've connected all the dots together. But if you make a mistake, all the lines will disappear and you'll have to start from the very first dot again.



◁ **Dots and lines**
The dots appear at random positions on the screen. When you click on the correct dot, a line will be drawn between it and the previous dot you clicked on.



Each dot has a number label under it.

When you click on the correct dot, a line is drawn between it and the last dot you clicked on.

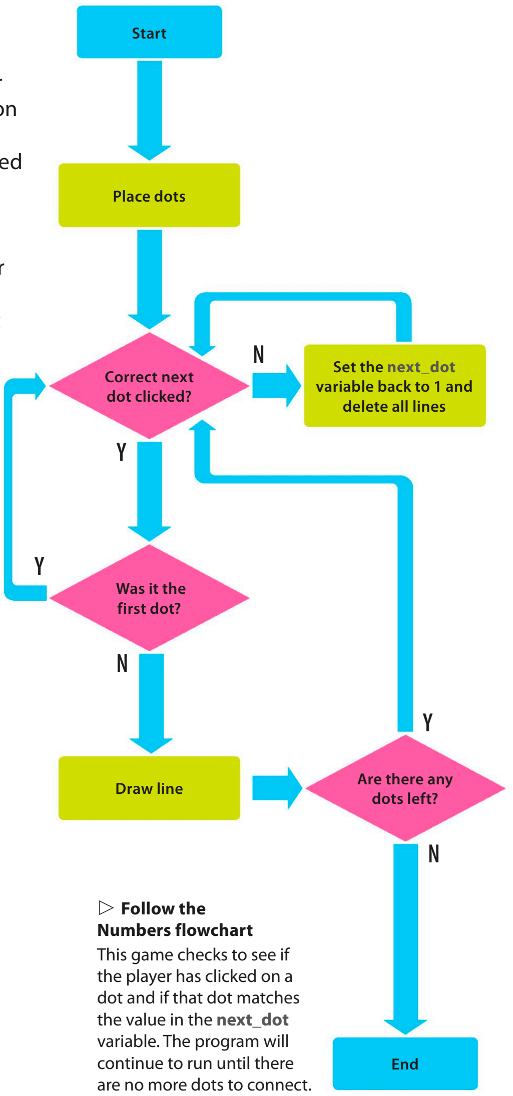
You can change the background to any color you like.

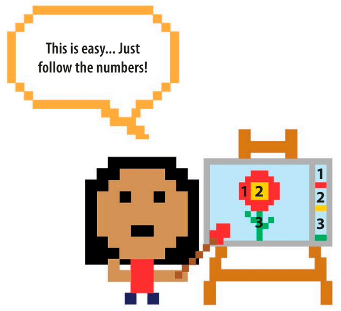
◁ **Connect the dots**
Every time you run this game, the program uses a loop to draw the dots at different positions on the screen.

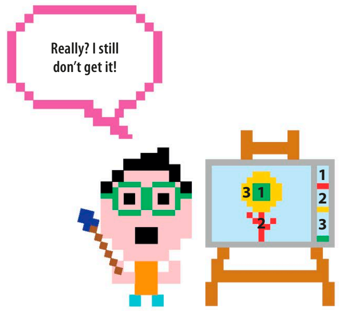
① network_basic

How it works

This game uses Python's **randint()** function to randomly choose x and y coordinates for each of the dots, and then places them all on the screen. It uses the **on_mouse_down()** function to know when the player has clicked on a dot. If the player clicks on the correct dot, and it's not the first dot, a line is drawn between the current dot and the previous dot. If the player clicks on the wrong dot, or clicks anywhere else on the screen, all the lines are deleted and the player has to start again. The game ends once the player has connected all the dots.


Start
Place dots
Correct next dot clicked?
N
Set the next_dot variable back to 1 and delete all lines
Y
Was it the first dot?
Y
N
Draw line
Are there any dots left?
Y
N
End





▷ **Follow the Numbers flowchart**
This game checks to see if the player has clicked on a dot and if that dot matches the value in the **next_dot** variable. The program will continue to run until there are no more dots to connect.

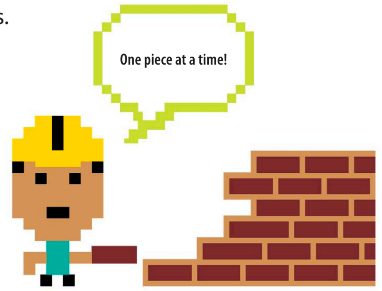
Let's get started

It's time to start building the game. Begin by importing the Python modules required for this game. Then write the functions to create the dots and the lines.

1 Set it up

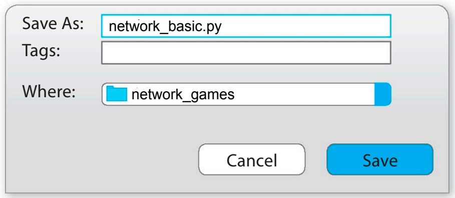
As usual we are going to create a separate folder to hold our network based games.

- Open your file explorer.
- Change directory to your `coderdojo_tramore`.
- Inside folder `coderdojo_tramore`, create a new subfolder called `network_games`.



2 Save the game

Save your file in the folder `network_games`. This should be inside your folder `coderdojo_tramore`. If you have difficulty finding this folder, please ask.



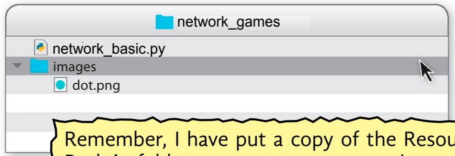
Save As: network_basic.py
Tags:
Where: network_games

3 Set up an image folder

This game uses one image for all the dots. You can create this folder using your file explorer or just by clicking on the images button in your Mu editor.

4 Put the image into the folder

Find the file called "dot.png" in the Python Games Resource Pack (**dk.com/computercoding**) and copy it into the images folder. Your folders should look something like this now.



network_games
network_basic.py
images
dot.png

Remember, I have put a copy of the Resource Pack in folder `coderdojo_trammore\resources`

5 Import a module

Now you're ready to start coding. Go back to your Mu file and type this line at the top.

```
from random import randint
```

This imports the **randint()** function from Python's Random module.

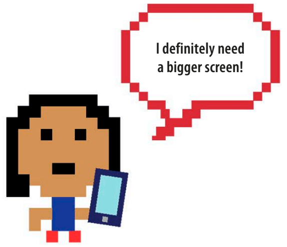
① network_basic

6 Set the screen size

Next you need to set the size of the screen for your game. Type these lines under the code from Step 5.

```
WIDTH = 400
HEIGHT = 400
```

This declares the global variables to set the screen size in pixels.



EXPERT TIPS

Global and local variables

There are two types of variables—local and global. A global variable can be used anywhere in your code. A local variable can only be used inside the function it was created in. To change a global variable in a function, just put the keyword **global** before its name.

7 Set up the lists

Now you need some lists to store all the dots, and also the lines that will be drawn to connect these dots. You'll need a variable to keep track of which dot should be clicked on next. Create these by typing this code.

```
HEIGHT = 400

dots = []
lines = []

next_dot = 0
```

These global lists will store the dots and the lines.

This global variable starts at **0** and tells the game which dot should be clicked on next.

8 Set up the Actors

It's time to set up the Actors. In this game, the ten dots are the Actors. Create these dots in a loop, giving each one a randomly chosen position and then adding it to the list of Actors. Type this code under what you typed in Step 7.

```
next_dot = 0

for dot in range(0, 10):
    actor = Actor("dot")
    actor.pos = randint(20, WIDTH - 20), \
    randint(20, HEIGHT - 20)
    dots.append(actor)
```

This will loop ten times.

This line will create a new Actor using the image of the dot in the images folder.

This will ensure that the dots appear at least 20 pixels away from the edge of the screen so the whole dot is shown.

Use a backslash character if you need to split a long line of code over two lines. It may fit on one in your file, though.

9 Draw the Actors

Now use the **draw()** function to display the dots and their number labels on the screen. The function **screen.draw.text()** expects a string as an input, but since the value stored in **number** is an integer, you need to use the **str()** function to convert it into a string. Add this code below the commands from Step 8.

```
    dots.append(actor)

def draw():
    screen.fill("black")
    number = 1
    for dot in dots:
        screen.draw.text(str(number), \
                         (dot.pos[0], dot.pos[1] + 12))
        dot.draw()
        number = number + 1
```

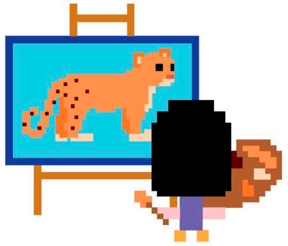
This sets the background color to black.

This creates a variable to keep track of the current number label.

These lines draw each dot on the screen along with a number label.

10 Draw the lines

Next add this code to the end of the **draw()** function to draw the lines. Until the player clicks on the first two dots, the lines list will remain empty, so the function won't draw any lines on the screen.



```
        number = number + 1
    for line in lines:
        screen.draw.line(line[0], line[1], (100, 0, 0))
```

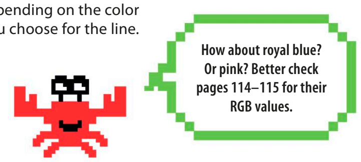
EXPERT TIPS

Line function

This function draws a line between two points on the screen—starting at point x and ending at point y. You can change the color of the line to red (R), green (G), blue (B), or even a mix of all three (RGB). Create a color by assigning values between 0 (none of the color) and 255 (the maximum amount of the color). For example, (0, 0, 100) sets the color of the line to blue. You can use some colors by typing in their names, but RGB values let you use lots of different shades.

```
screen.draw.line(x, y, (0, 0, 100))
```

These numbers can change depending on the color you choose for the line.


How about royal blue? Or pink? Better check pages 114–115 for their RGB values.

① network_basic

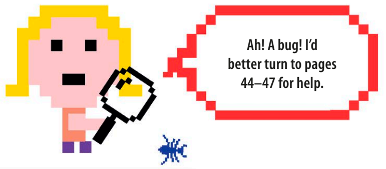
11 Test Your Code

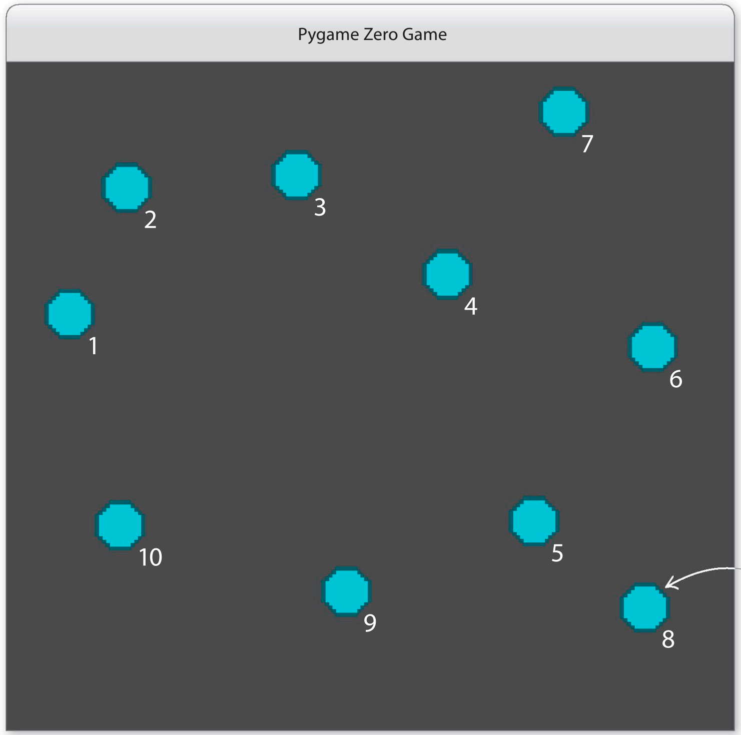
Let's test the code that you have written so far.

Remember, it is always a good idea to check code as often as possible both to see if what you have just implemented is correct and that you have not broken some earlier code by your most recent changes.

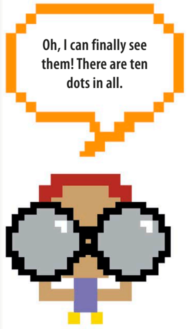
12 What do you see?

If the program runs successfully, you should see a screen like the one below. Your dots will probably be in a slightly different place, though. If your screen looks completely different, or if you get an error message, go through your code carefully to see if you've made any mistakes.





The position of the dots will change each time you run the code.



13 Add a new function

When you ran the program just then, you probably noticed that nothing happened when you clicked on the dots. To fix this, add the **on_mouse_down(pos)** function under the code from Step 10.

```
def on_mouse_down(pos):
    global next_dot
    global lines
```

You have to add this code to let the function change the values of the global variables **next_dot** and **lines**.

14 Connect the dots

You now need to make the dots respond to the mouse clicks. Add these lines under **def on_mouse_down(pos)** from Step 13.

```
    global lines
    if dots[next_dot].collidepoint(pos):
        if next_dot:
            lines.append((dots[next_dot - 1].pos, dots[next_dot].pos))
        next_dot = next_dot + 1
    else:
        lines = []
        next_dot = 0
```

This line checks if the player has clicked on the next dot in the sequence.

This line checks if the player has already clicked on the first dot.

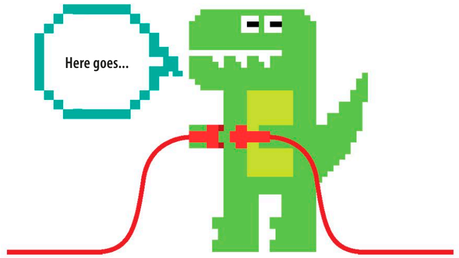
This draws a line between the current dot and the previous one.

This sets **next_dot** to the next number.

If the player clicks on the wrong dot, this sets the **next_dot** back to the first one and deletes all the lines.

15 Time to connect

And it's done! Now that you've finished writing the code, save it and run it from the command line to start playing. Don't forget, you need to connect all the dots as fast as you can!



EXPERT TIPS

Collisions

You can use the **collidepoint()** function to check if the position of the mouse click matches the position of an Actor.

This creates an Actor with the dot image.

This passes the position of the mouse click to the **on_mouse_down()** function.

```
dot = Actor("dot")

def on_mouse_down(pos):
    if dot.collidepoint(pos):
        print("Ouch")
```

If the mouse click position and the dot position match, "Ouch" is printed in the shell.

① network_basic

Hacks and tweaks

Try out the following ideas to make Follow the Numbers a bit more challenging and even more fun.

△ **More dots**
You can add more dots to the game to make it more challenging. Remember the loop in Step 8 that creates ten dots? Can you modify the range to create some more?

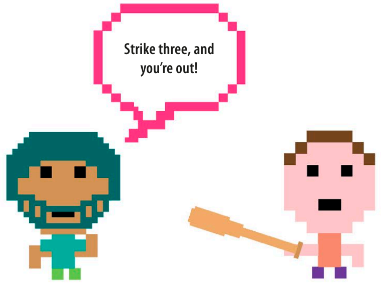
Set up a variable to keep track of how many dots each level has.

Define a function that adds two dots to the **dots** list.

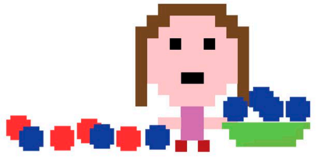
```
number_of_dots = 10

def next_level:

if next_dot == number_of_dots - 1:
```

What does your program need to do when increasing the level?

△ **Level up**
You could add levels so the game gets harder each time you complete a sequence. Each level could have two more dots than the last. Try defining a **next_level**() function to do this. This code will help you get started.

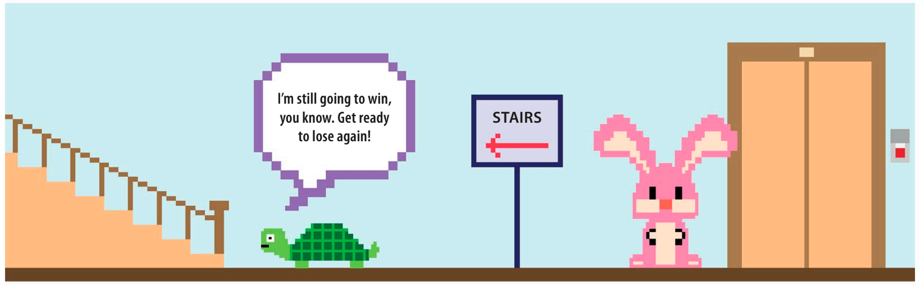


△ **No more chances**
At the moment, the player has an unlimited number of attempts to connect the dots together. Try changing the code so that the game ends if the player makes a mistake. You could even add a "Game Over!" message to your code. If you do this, remember to clear everything else off the screen first.



△ **Multiple sets of dots**
To make the game more challenging, you could add another set of dots. There's a red dot in the Hacks and tweaks section of the Resource Pack. You'll need to think about the following things to tweak the game:

- Create a separate list for the red dots.
- Create a separate list for blue lines to connect the red dots.
- Create a **next_dot** variable for the red dots.
- Set up the red dots at the start.
- Draw the red dots and blue lines.
- Check if the next red dot has been clicked.



▷ **In record time**
You can use the system clock to time how long it takes a player to connect all the dots. You could then try to beat your friends' times! To time the game, you'll need to use the **time**() function. Once the game is complete, you can display the final time taken on the screen. Why not try placing the clock in the corner? Remember to use **str**() to cast the message into a string. You can check Step 9 of the game if you need to remind yourself how to do this. At the moment, though, the **draw**() function is only called when the player clicks the mouse, so the clock would only update after each mouse click. To fix this, add this code. This function is called 60 times a second. Each call also calls the **draw**() function, so the clock stays up to date.

```
from time import time
```

Put this code at the top of your program to use the Time module.

```
def update():
    pass
```

You don't need to replace **pass** with any actual code.

EXPERT TIPS
time()

The **time**() function might give you an unexpected result. It calculates the time that's passed since an "epoch," which is the date an operating system considers to be the "start of time." Windows machines will tell you how many seconds have passed since January 1, 1601! You can use this simple calculation below to work out how long it actually took the player to complete the game.

```
total_time = end_time - start_time
```

This calculates the total time elapsed.

EXPERT TIPS
round()

The **time**() function calculates time to lots of decimal places. You can use the **round**() function to round it to a certain number of decimal places, which will make it easier to read. **round**() takes two parameters—the number to round up or down and the number of decimal places to shorten it to.

```
>>> round(5.75, 1)
5.8
```

This is the number of decimal places you want to round it to.

This is the number you want to round up.

1 Refactor game

Create a new file and copy the contents of `network_basic.py` into it. Save new file as `network_refactored.py` in your `network_games` folder. Then make the following changes.

```python
for k in range(10):
    actor = Actor("dot")
    actor.pos = randint(20, WIDTH - 20), randint(20, HEIGHT - 20)
    dots.append(actor)
```

It is important to think carefully about the names we give to our data in our programs. Here `dot` is better because it is more specific and more informative than `actor`.

Change identifier name from `actor` to `dot`

```python
for k in range(10):
    dot = Actor("dot")
    dot.pos = randint(20, WIDTH - 20), randint(20, HEIGHT - 20)
    dots.append(dot)
```

```python
def draw():
    screen.fill("black")

    number = 1
    for dot in dots:
        screen.draw.text(str(number), (dot.x, dot.y + 12))
        number = number + 1
        dot.draw()

    for line in lines:
        screen.draw.line(line[0], line[1], (255, 255, 0))
```

```python
def draw():
    screen.fill("black")

    for n, dot in enumerate(dots):
        screen.draw.text(str(n+1), (dot.x, dot.y + 12))
        dot.draw()

    for line in lines:
        start, end = line
        screen.draw.line(start, end, (255, 255, 0))
```

In function `draw` I have made two changes:

- used **`enumerate`** to count over the dots, so don't need variable `number` and **`enumerate`** will be responsible for remembering to update `n` and not me! But note that **`enumerate`** starts counting at zero.
- the identifier `line` is a list of two points, rather than referring to the two points by number, it is nicer and clearer to give them names — I have used `start` and `end`.

Make the above changes.

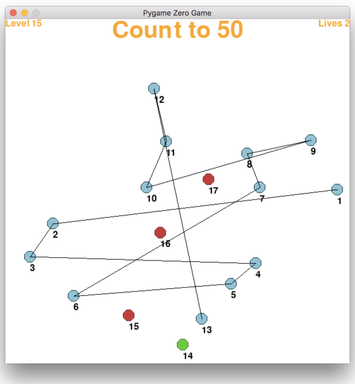
2 More advanced games

2.1 Count to 50

Copy the contents of `network_refactored.py` into a new file and save it as `count_to_50.py` in the `network_games` folder.

We can improve the current game in a number of ways — one way is to develop the following 'Count to 50' game:

- Screen starts with three (level 3) dots, and with each level the number of dots increase by one.
- To help the player the dots are coloured as follows: dots already joined together are blue, the next dot to click on is green, and all other dots are red.
- Players have 3 lives, they lose a life every time they click on an incorrect dot.
- The game is timed and players have to race through the levels before the time runs out.



- Doing all of the levels from 3 to 50 will require a lot of clicks (will require 5044 clicks!) so we want some way for players to "hyper-jump" a few levels occasionally or gain time bonus by say getting to a dot particularly quickly. Have a think about what you could do here and talk to us in terms of how to implement your ideas.
- One small detail remains — when you currently generate the random positions for the dots sometimes one dot is (partially) covered by another. We don't want that since then the player cannot click on the correct dot. There is an easy fix for that, but rather than us telling you, have a think about it first.

2.2 Make a Total

Copy the contents of `network_refactored.py` into a new file and save it as `make_a_total.py` in the `network_games` folder.

- Instead of counting up what about displaying at random number and the player then needs to click on dots to together sum up to the displayed figure — they get extra points for using the fewest number of dots.
- In this version all of the dots will be blue as the player will need to decide which numbers that want to use to make the required total. If the dots clicked on go above the total then the level restarts and they lose a life,
- Again, to add pressure, have the levels timed so you lose a life if you don't get the total within the given time limit.
- One of the change that I would do to make writing this game easier is to store in each dot its label, using something like the following:

```
dot.label = n
```

- Or what about going *extreme total* and have both green and red dots — to get a total you add the green dots and subtract the red? That will make life much harder!