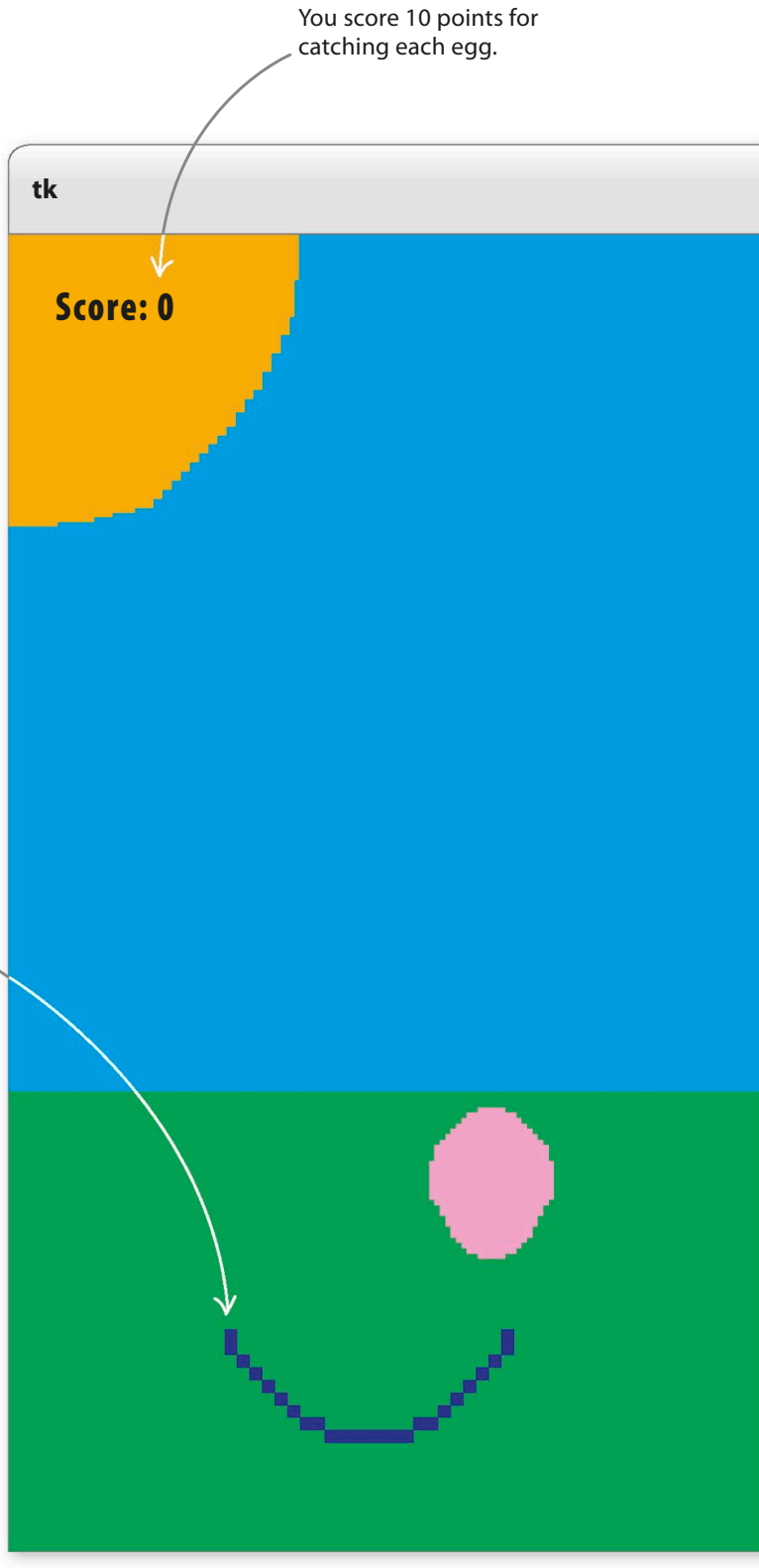
Egg Catcher

This game will test your concentration and the speed of your reflexes. Don't crack under pressure—just catch as many eggs as you can to get a high score. Challenge your friends to see who is the champion egg catcher!

What happens

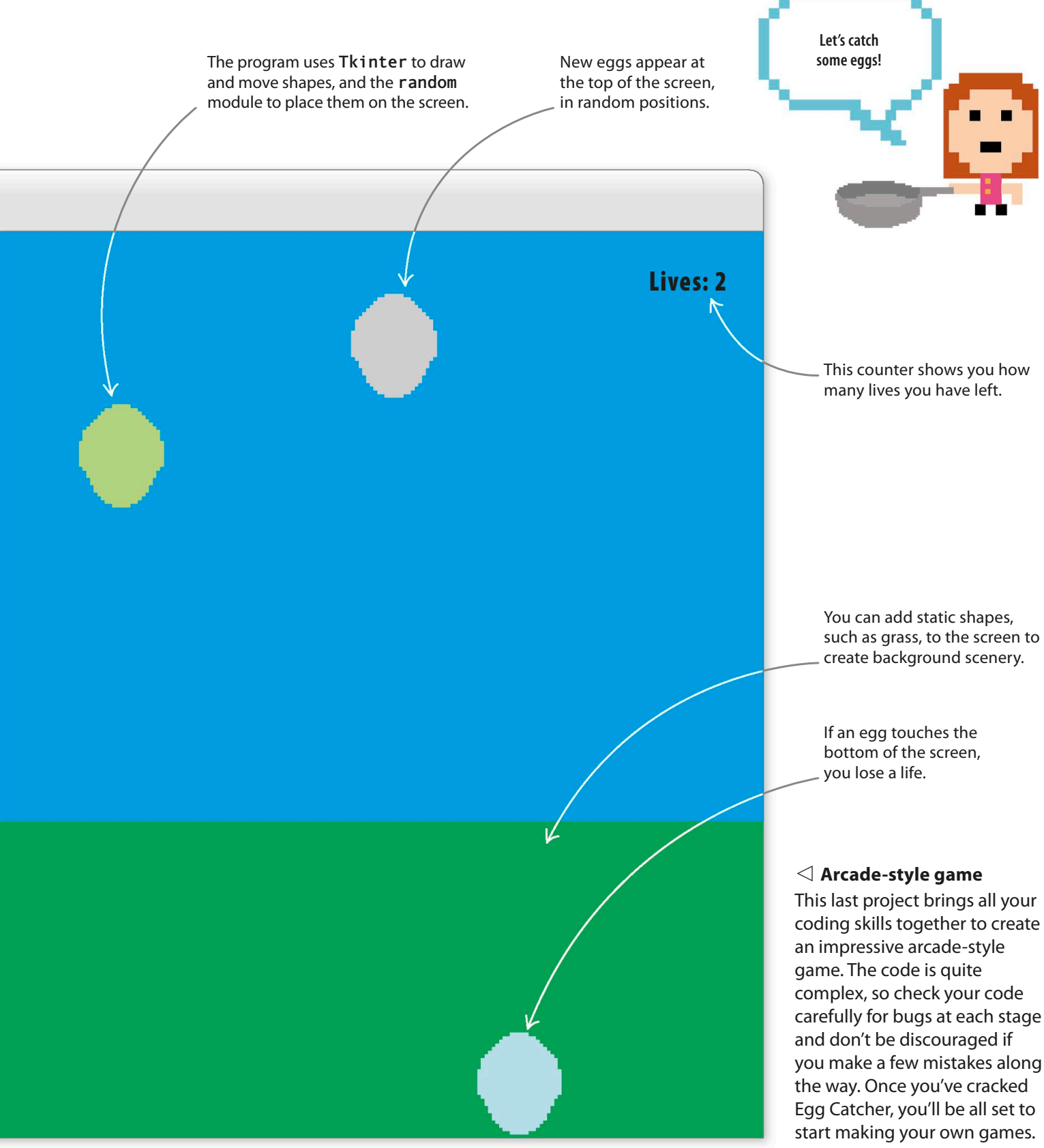
Move the catcher along the bottom of the screen to catch each egg before it touches the ground. When you scoop up an egg you score points, but if you drop an egg you lose a life. Beware: the more eggs you catch, the more frequently new eggs appear at the top of the screen and the faster they fall. Lose all three lives and the game ends.



EXPERT TIPS

Timing

The timing of the action on the screen is important. At first, a new egg is only added every 4 seconds; otherwise, there would be too many eggs. Initially, the eggs move down a little every half second. If the interval was smaller, the game would be too hard. The program checks for a catch once every tenth of a second—any slower, and it might miss it. As the player scores more points, the speed and number of the eggs increases to make the game more challenging.

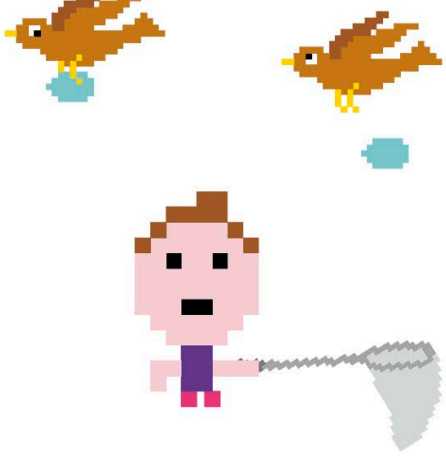


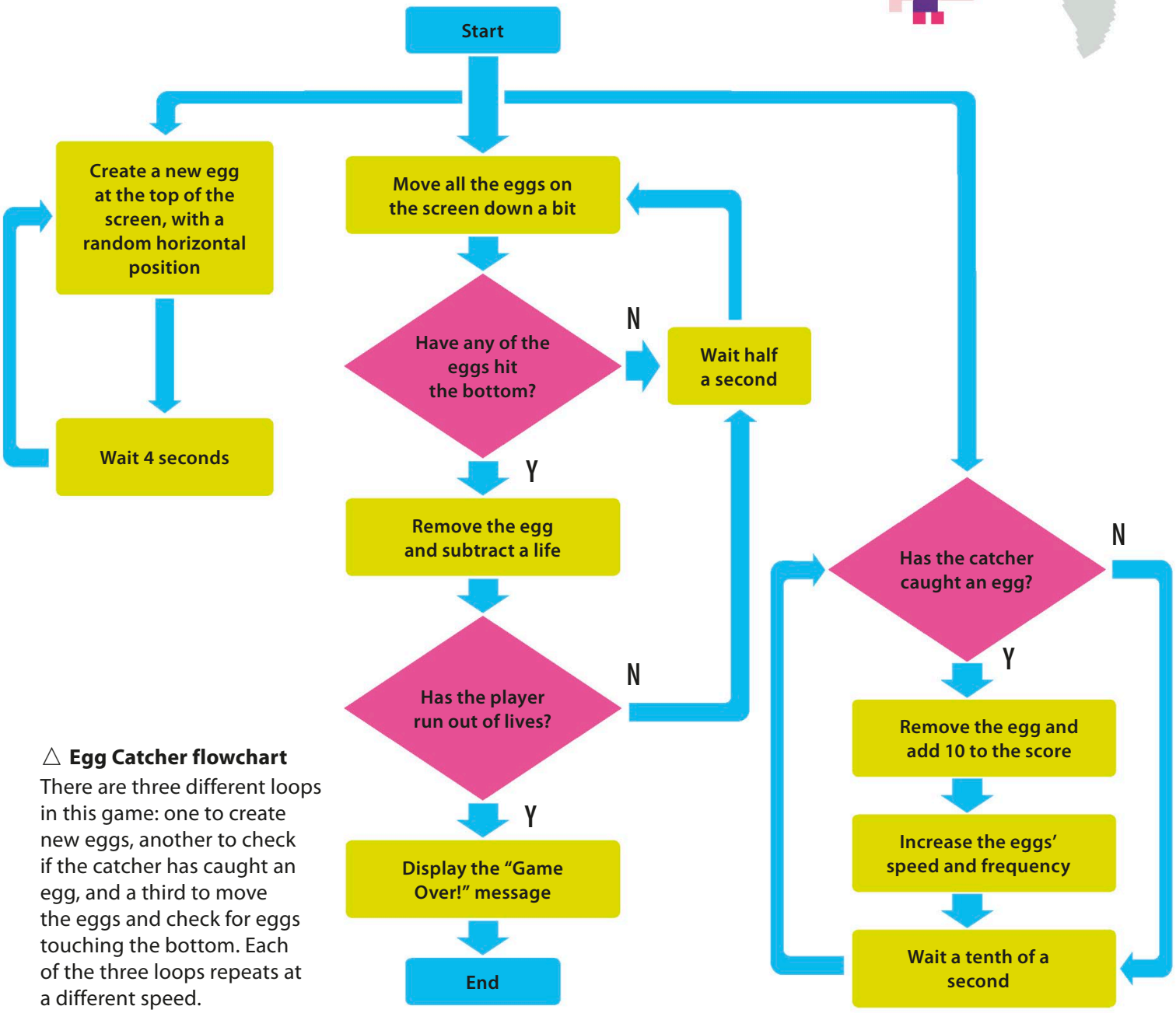
◁ **Arcade-style game**

This last project brings all your coding skills together to create an impressive arcade-style game. The code is quite complex, so check your code carefully for bugs at each stage and don't be discouraged if you make a few mistakes along the way. Once you've cracked Egg Catcher, you'll be all set to start making your own games.

How it works

Once the background is created, the eggs gradually move down the screen, which creates the illusion that they are falling. Using loops, the code continually checks the coordinates of the eggs to see if any have hit the bottom or been caught in the catcher. When an egg is caught or dropped, it is deleted and the program adjusts the score or the number of remaining lives.





△ **Egg Catcher flowchart**
There are three different loops in this game: one to create new eggs, another to check if the catcher has caught an egg, and a third to move the eggs and check for eggs touching the bottom. Each of the three loops repeats at a different speed.

Setting up

First you'll import the parts of Python that you need for this project. Then you'll set things up that so that you're ready to write the main functions for the game.

1 Open IDLE and create a new file. Save it as "egg_catcher.py".

2 **Import the modules**

Egg Catcher uses three modules: `itertools` to cycle through some colors; `random` to make the eggs appear in random places; and `Tkinter` to animate the game by creating shapes on the screen. Type these lines at the top of your file.

```
from itertools import cycle
from random import randrange
from tkinter import Canvas, Tk, messagebox, font
```

The code only imports the parts of the modules that you need.

3 **Set up the canvas**

Add this code beneath the import statements. It creates variables for the height and width of the canvas, then uses them to create the canvas itself. To add a bit of scenery to your game, it draws a rectangle to represent some grass and an oval to represent the sun.

```
from tkinter import Canvas, Tk, messagebox, font

canvas_width = 800
canvas_height = 400

root = Tk()
c = Canvas(root, width=canvas_width, height=canvas_height, \
background='deep sky blue')
c.create_rectangle(-5, canvas_height - 100, canvas_width + 5, \
canvas_height + 5, fill='sea green', width=0)
c.create_oval(-80, -80, 120, 120, fill='orange', width=0)
c.pack()
```

This creates a window.

The canvas will be sky blue and measure 800 x 400 pixels.

Use a backslash character if you need to split a long line of code over two lines.

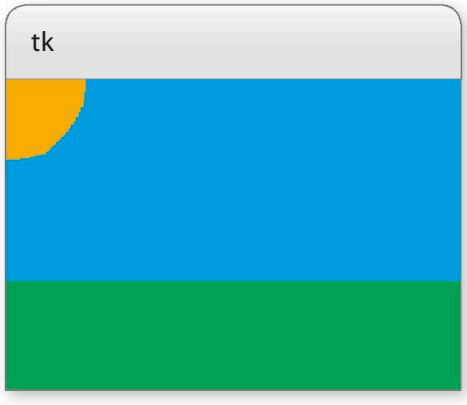
This creates the grass.

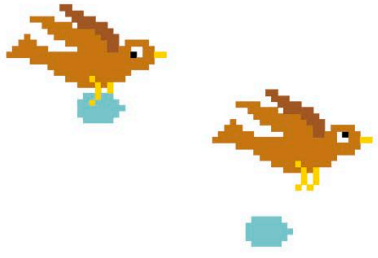
The `pack()` function tells the program to draw the main window and all of its contents.

This line creates the sun.

4 **See your canvas**

Run the code to see how the canvas looks. You should see a scene with green grass, a blue sky, and a bright sun. If you feel confident, try to make your own scenery with shapes of different colors or sizes. You can always go back to the code above if you run into problems.





Now make some variables to store the colors, width, and height of the eggs. You'll also need variables for the score, the speed of the falling eggs, and the interval between new eggs appearing on the screen. The amount they are changed by is determined by the `difficulty_factor`—a lower value for this variable actually makes the game harder.

```
c.pack()

color_cycle = cycle(['light blue', 'light green', 'light pink', 'light yellow', 'light cyan'])
egg_width = 45
egg_height = 55
egg_score = 10
egg_speed = 500
egg_interval = 4000
difficulty_factor = 0.95
```

The `cycle()` function allows you to use each color in turn.

You score 10 points for catching an egg.

A new egg appears every 4,000 milliseconds (4 seconds).

This is how much the speed and interval change after each catch (closer to 1 is easier).

6 Set up the catcher

Next add the variables for the catcher. As well as variables for its color and size, there are four variables that store the catcher's starting position. The values for these are calculated using the sizes of the canvas and the catcher. Once these have been calculated, they are used to create the arc that the game uses for the catcher.

Don't forget to save your work.

```
difficulty_factor = 0.95

catcher_color = 'blue'
catcher_width = 100
catcher_height = 100
catcher_start_x = canvas_width / 2 - catcher_width / 2
catcher_start_y = canvas_height - catcher_height - 20
catcher_start_x2 = catcher_start_x + catcher_width
catcher_start_y2 = catcher_start_y + catcher_height

catcher = c.create_arc(catcher_start_x, catcher_start_y, \
                       catcher_start_x2, catcher_start_y2, start=200, extent=140, \
                       style='arc', outline=catcher_color, width=3)
```

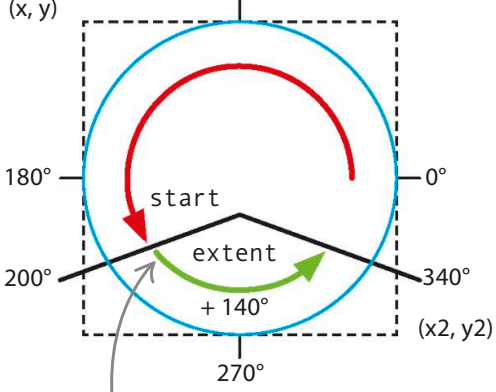
This is the height of the circle that is used to draw the arc.

These lines make the catcher start near the bottom of the canvas, in the center of the window.

Start drawing at 200 degrees on the circle.

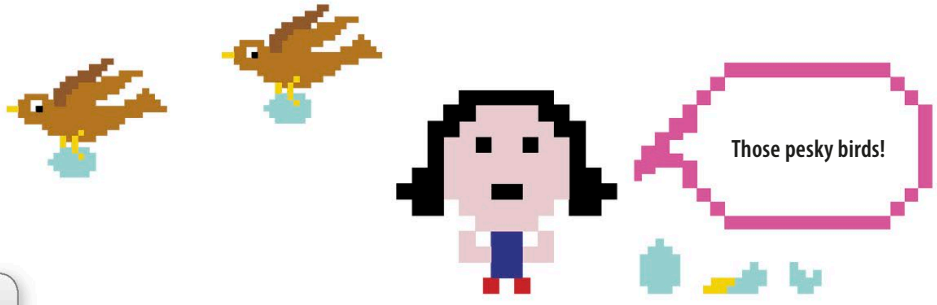
Draw for 140 degrees.

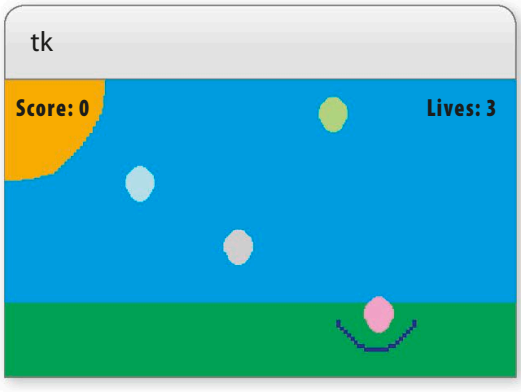
Draw the catcher.



A whole circle is 360°. The code starts drawing the arc just over half way around the circle, at 200°.

You use an arc to represent the catcher. An arc is one part of a whole circle. `Tkinter` draws circles inside an invisible box. The first two `catcher_start` coordinates (x and y) plot where one corner of the box should be. The second two coordinates (x2 and y2) plot the position of the box's opposite corner. The `create_arc()` function has two parameters, both given in degrees (°), that say where in the circle to draw the arc: `start` says where to start drawing, while `extent` is how many degrees to draw before stopping.





7 Score and lives counters

Add this code under the lines that set up the catcher. It sets the starting score to 0 and creates the text that shows the score on the screen. It also sets the remaining lives to three and displays this number. To check if the code is working, add `root.mainloop()` right at the end and then run the code. Once you've checked, remove this line—you'll add it again later when it's needed.

```
catcher = c.create_arc(catcher_start_x, catcher_start_y, \
                       catcher_start_x2, catcher_start_y2, start=200, extent=140,
                       style='arc', outline=catcher_color, width=3)

game_font = font.nametofont('TkFixedFont')
game_font.config(size=18)

score = 0
score_text = c.create_text(10, 10, anchor='nw', font=game_font, fill='darkblue', \
                           text='Score: ' + str(score))

lives_remaining = 3
lives_text = c.create_text(canvas_width - 10, 10, anchor='ne', font=game_font, \
                           fill='darkblue', text='Lives ' + str(lives_remaining))
```

This line selects a cool computer-style font.

You can make the text larger or smaller by changing this number.

The player gets three lives.

Falling, scoring, dropping

You've completed all the setup tasks, so it's time to write the code that runs the game. You'll need functions to create the eggs and make them fall, and some more functions to handle egg catches and egg drops.

8

Add this code. A list keeps track of all the eggs on the screen. The **`create_egg()`** function decides the coordinates of each new egg (the x coordinate is always randomly selected). Then it creates the egg as an oval and adds it to the list of eggs. Finally, it sets a timer to call the function again after a pause.

```
lives_text = c.create_text(canvas_width - 10, 10, anchor='ne', font=game_font, fill='darkblue', \
                           text='Lives: ' + str(lives_remaining))

eggs = []

def create_egg():
    x = randrange(10, 740)
    y = 40
    new_egg = c.create_oval(x, y, x + egg_width, y + egg_height, fill=next(color_cycle), width=0)
    eggs.append(new_egg)
    root.after(egg_interval, create_egg)
```

This is a list to keep track of the eggs.

Pick a random position along the top of the canvas for the new egg.

This line of code creates the oval.

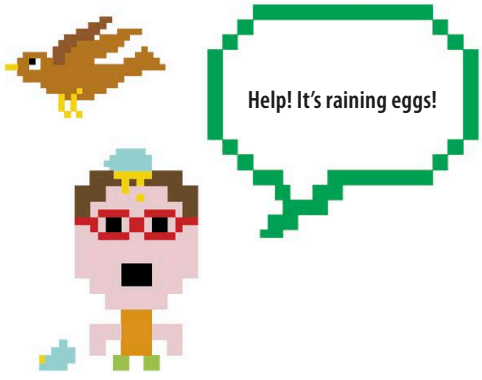
The shape is added to the list of eggs.

Call this function again after the number of milliseconds stored in **`egg_interval`**.

9

Move the eggs

After creating the eggs, add the next function, **`move_eggs()`**, to set them in motion. It loops through the list of all the eggs on screen. For each egg, the y coordinate is increased, which moves the egg down the screen. Once the egg is moved, the program checks whether it has hit the bottom of the screen. If it has, the egg has been dropped and the **`egg_dropped()`** function is called. Finally, a timer is set to call the **`move_eggs()`** function again after a short pause.




```
    root.after(egg_interval, create_egg)

def move_eggs():
    for egg in eggs:
        (egg_x, egg_y, egg_x2, egg_y2) = c.coords(egg)
        c.move(egg, 0, 10)
        if egg_y2 > canvas_height:
            egg_dropped(egg)
    root.after(egg_speed, move_eggs)
```

Loop through all the eggs.

This line gets each egg's coordinates.

The egg drops down the screen 10 pixels at a time.

Is the egg at the bottom of the screen?

If so, call the function that deals with dropped eggs.

Call this function again after the number of milliseconds stored in **`egg_speed`**.

10 Oops—egg drop!

Next add the `egg_dropped()` function after `move_eggs()`. When an egg is dropped, it is removed from the list of eggs and then deleted from the canvas. A life is deducted using the `lose_a_life()` function, which you'll create in Step 11. If losing a life means there are no lives left, the "Game Over!" message is shown.

```
    root.after(egg_speed, move_eggs)

def egg_dropped(egg):
    eggs.remove(egg)
    c.delete(egg)
    lose_a_life()
    if lives_remaining == 0:
        messagebox.showinfo('Game Over!', 'Final Score: ' \
                            + str(score))
        root.destroy()
```

The egg is removed from the eggs list.

The egg disappears from the canvas.

This line calls the `lose_a_life()` function.

If no lives are left, tell the player that the game is over.

The game ends.

11 Lose a life

Losing a life simply involves subtracting a life from the `lives_remaining` variable and then displaying the new value on the screen. Add these lines after the `eggs_dropped()` function.

```
        root.destroy()

def lose_a_life():
    global lives_remaining
    lives_remaining -= 1
    c.itemconfigure(lives_text, text='Lives: ' \
                    + str(lives_remaining))
```

This variable needs to be global, as the function will modify it.

The player loses a life.

This line updates the text that shows the remaining lives.

12 Check for a catch

Now add the `check_catch()` function. An egg is caught if it's inside the arc of the catcher. To find out if you've made a catch, the `for` loop gets the coordinates of each egg and compares them with the catcher's coordinates. If there's a match, the egg is caught. Then it's deleted from the list, removed from the screen, and the score is increased.

```
    c.itemconfigure(lives_text, text='Lives: ' + str(lives_remaining))

def check_catch():
    (catcher_x, catcher_y, catcher_x2, catcher_y2) = c.coords(catcher)
    for egg in eggs:
        (egg_x, egg_y, egg_x2, egg_y2) = c.coords(egg)
        if catcher_x < egg_x and egg_x2 < catcher_x2 and catcher_y2 - egg_y2 < 40:
            eggs.remove(egg)
            c.delete(egg)
            increase_score(egg_score)
    root.after(100, check_catch)
```

Get the coordinates of the catcher.

Get the coordinates of the eggs.

Is the egg inside the catcher horizontally and vertically?

Increase the score by 10 points.

Call this function again after 100 milliseconds (one-tenth of a second).

13 First the score is increased by the value of the `points` parameter. Next the new speed and interval of the eggs are calculated by multiplying their values by the difficulty factor. Finally, the text on the screen is updated with the new score. Add this new function beneath `check_catch()`.


```
    root.after(100, check_catch)

def increase_score(points):
    global score, egg_speed, egg_interval
    score += points
    egg_speed = int(egg_speed * difficulty_factor)
    egg_interval = int(egg_interval * difficulty_factor)
    c.itemconfigure(score_text, text='Score: ' + str(score))
```

Add to the player's score.

This line updates the text that shows the score.

Catch those eggs!

Now that you've got all the shapes and functions needed for the game, all that's left to add are the controls for the egg catcher and the commands that start the game.

14 **Set up the controls**

The `move_left()` and `move_right()` functions use the coordinates of the catcher to make sure it isn't about to leave the screen. If there's still space to move to, the catcher shifts horizontally by 20 pixels. These two functions are linked to the left and right arrow keys on the keyboard using the `bind()` function. The `focus_set()` function allows the program to detect the key presses. Add the new functions beneath the `increase_score()` function.

```
    c.itemconfigure(score_text, text='Score: \
                    ' + str(score))

def move_left(event):
    (x1, y1, x2, y2) = c.coords(catcher)
    if x1 > 0:
        c.move(catcher, -20, 0)

def move_right(event):
    (x1, y1, x2, y2) = c.coords(catcher)
    if x2 < canvas_width:
        c.move(catcher, 20, 0)

c.bind('<Left>', move_left)
c.bind('<Right>', move_right)
c.focus_set()
```

Has the catcher reached the left-hand wall?

If not, move the catcher left.

Has the catcher reached the right-hand wall?

If not, move the catcher right.

These lines call the functions when the keys are pressed.

15 The three looping functions are started using timers. This ensures they aren't run before the main loop starts. Finally, the **`mainloop()`** function starts the **`Tkinter`** loop that manages all your loops and timers. All finished – enjoy the game, and don't let those eggs smash!

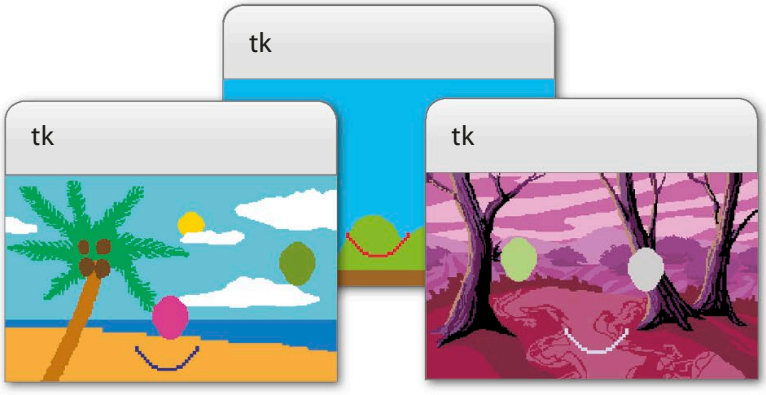
```
c.focus_set()

root.after(1000, create_egg)
root.after(1000, move_eggs)
root.after(1000, check_catch)
root.mainloop()
```

The three game loops begin after a slight pause of 1,000 milliseconds (1 second).

This line starts the main **`Tkinter`** loop.

Hacks and tweaks

To make the game look even better, you can try adding some cool scenery of your own. Fun sounds and music are another great way to make the game more exciting.



EXPERT TIPS

Installing modules

Some of the most useful Python modules—such as **`Pygame`**—aren't included as part of the standard Python library. If you would like to use any of these other modules, you'll need to install them first. The best place to look for instructions on how to install a module is the module's website. There are instructions and tips at **https://docs.python.org/3/installing/**.

◁ Set the scene

`Tkinter` allows custom images to be used as backgrounds for a canvas. If your file is a GIF, you can use **`tkinter.PhotoImage`** to load the file. If your image is a different format, you might want to look into **`Pillow`**—a helpful image-handling module.

▷ Make some noise

To really bring the game to life, add background music or sound effects for catching an egg or losing a life. The module to use for adding sounds is **`pygame.mixer`**. Remember, **`pygame`** is not a standard Python module, so you'll need to install it first. You'll also need to have a copy of the sound file you want to play, which you should place in the same folder as your code file. Once that's in place, playing a sound only takes a few lines of code.

```
import time


from pygame import mixer

mixer.init()
beep = mixer.Sound("beep.wav")
beep.play()
time.sleep(5)
```

Get the mixer ready to play sounds.

Tell the mixer which sound to play.

Play the sound.

Keep the program running long enough to hear it.