

Collector Games

Introduction

Collector games are based around the idea of the player moving around the screen trying to collect coins as quickly as possible.

As in our fruit ninja game we will develop multiple versions as we add more and more features. The features for this game are similar to what you did for the Fruit Ninja games to test how well you understood the code you implemented there.

① **Basic coin collector game** collector_basic.py

Our starting version of the game consists of coins appearing randomly on the screen and the player has to move the character to collect the coins before the limited time runs out.

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

② **Adding sounds** collector_sound.py

Go to www.zapsplat.com, and find some sounds that we can use when you collect a coin, perhaps a slot-machine ring would work.



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 **Making the game more playable** collector_levels.py

Currently the game just runs down a clock and your score is how many coins selected. The problem here is that there is no real goal — other than collecting as many as possible. A nicer version would be to have multiple levels, and in each level the player must pick up, say 5 coins to win, but the time allocated gets shorter and shorter.

④ **Varying targets** collector_targets.py

We could have multiple targets — some of which are worth more than others.

⑤ **Disappearing targets** collector_vanishing_targets.py

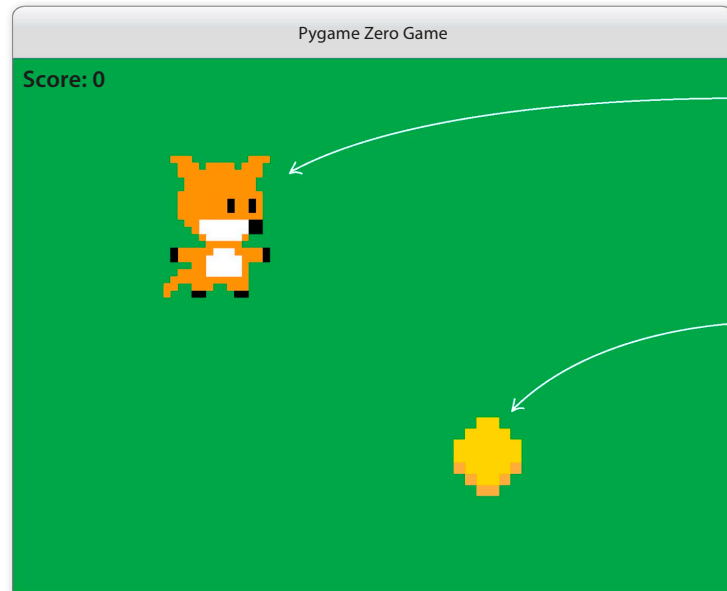
We could have targets that disappear if they are not picked within a (small) time interval.

How to build Coin Collector

Help a crafty fox collect as many coins as possible before the time runs out. The more coins you get, the higher your score. Be quick! You only have a few seconds to collect them.

What happens

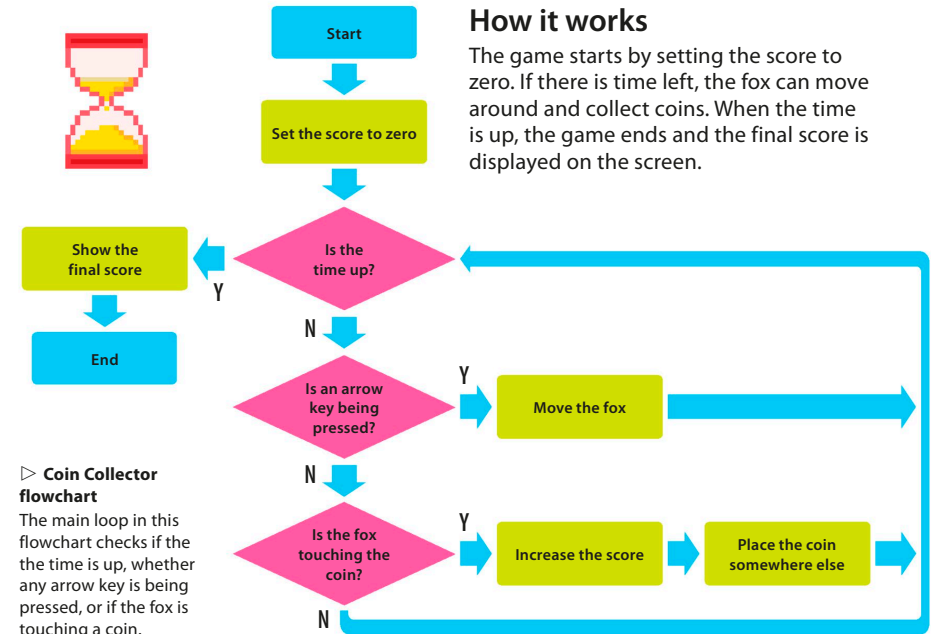
A fox and a coin appear on the screen. You use the arrow keys to move the fox toward the coin. When the fox touches the coin, you get ten points, and another coin appears somewhere else. The game ends after seven seconds and the final score is displayed.



◀ **Need for speed**
This time-based game tests your reaction speed. How fast can you move the fox to grab the coins before the time's up?

How it works

The game starts by setting the score to zero. If there is time left, the fox can move around and collect coins. When the time is up, the game ends and the final score is displayed on the screen.



Getting started

Follow these steps to build the game. First set up a new file and import the relevant modules. Then draw the Actors and define the functions to run the game. Good luck!




1 Setup Game

In the mu-editor,

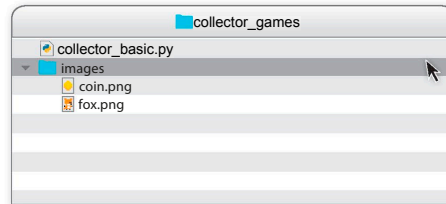
- Click on the new file button,
- Click on the save file button,
- Change folder to `coderojo_tramore`
- Create a new folder called `collector_games`
- Save game in new folder, using name `collector_basic`

2 Set up an image folder

This game uses two images—a fox and a coin. Within your collector_games folder, we need to create a new folder called images. The Mu editor will do this for us, when you click on the images button, .

3 Put the images into the folder

Find the files called “coin.png” and “fox.png” in the Python Games Resource Pack (dk.com/computercoding). Copy them both into the images folder. Your folders should look like this now.

**4 Get coding**

Now you're ready to start coding. This game works in a similar way as Shoot the Fruit, so you'll be able to reuse some of the code from that game. Begin by setting the size of the playing area. Type this code at the top of your file.

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

This code will make the game screen 400 pixels tall and 400 pixels wide.

5 Setting the score

Now, let's set the score to zero to begin with. You'll need to use a variable to do this. Type the code shown in black below.

```
WIDTH = 400
HEIGHT = 400
score = 0
```

This sets up a variable called score.

6 Game over?

You also need a Boolean variable (a variable whose value can either be True or False) to tell Pygame Zero if the game is over or not. At this stage, set the variable to False.

```
WIDTH = 400
HEIGHT = 400
score = 0
game_over = False
```

7 Introducing the Actors

This game will feature two Actors—a fox and a coin. To create them and set their positions, add these lines of code under what you typed in Step 6.

The coin is positioned 200 pixels along from the top left and 200 pixels down.

```
fox = Actor("fox")
fox.pos = 100, 100
coin = Actor("coin")
coin.pos = 200, 200
```

This line uses the fox.png file in the images folder to create the fox Actor.

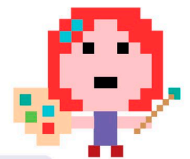
8 Time to draw

Now you need to use the draw() function to display the Actors on the screen, change the background color, and display the score. Type in this code to do these.

These lines draw the fox and coin on the screen.

```
coin.pos = 200, 200

def draw():
    screen.fill("green")
    fox.draw()
    coin.draw()
    screen.draw.text("Score: " + str(score), color="black", topleft=(10, 10))
```



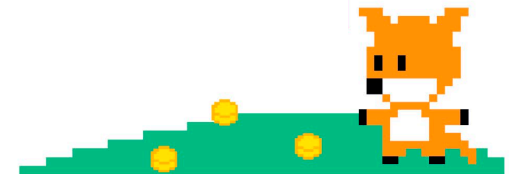
This line will display the score in the top-left corner of the screen.

9 Try it out

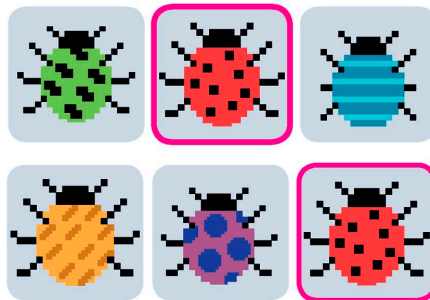
Now test the code you've written so far.

10 Did it work?

Did your game run? You should see the fox and coin on your screen, with the score in the top-left corner. You can't actually play the game yet, but it's a good idea to run your code frequently to check for bugs.

**LINGO****Patterns**

Lots of computer games follow patterns. Even though two games might have different characters, power-ups, or levels, their actual rules may be quite similar. Computer programmers often look for patterns in the programs they are building. If they spot a pattern, they can reuse some code from an existing program, making it easier and quicker to build the new program. This code is also less likely to have bugs because it will already have been tested.



11 Using placeholders

You need to write some more functions in order to finish the game. You can add function placeholders without having to define them right away by using the keyword `pass`. Type in this code to give yourself a template of the functions you'll need.

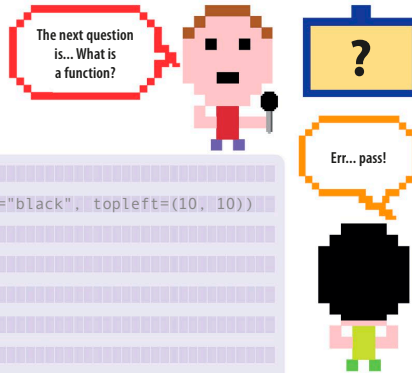
```
coin.draw()
screen.draw.text("Score: " + str(score), color="black", topleft=(10, 10))

def place_coin():
    pass

def time_up():
    pass

def update():
    pass
```

To get an idea of the code's structure, you can use placeholders for functions that you'll finish coding later.

**12 Importing randint()**

Now it's time to define these functions. The first one will use Python's built-in `randint()` function, so you need to import it into your program. Type this line at the very top of your code to import it.

```
from random import randint
```

Make sure you type this before all the code you've written so far.

14 Run the function

Remember, it's not enough just to define the function; you have to run it, too. Add this line of code to the very bottom of your game.

```
def update():
    pass

place_coin()
```

This will run the code you've saved in the `place_coin()` function.

13 Placing the coin

Next change the code in your `place_coin()` function. This function will place the coin in a random position on the screen. Delete `pass` and type in these commands.

```
def place_coin():
    coin.x = randint(20, (WIDTH - 20))
    coin.y = randint(20, (HEIGHT - 20))
```

The coin will be placed at least 20 pixels in from the sides of the screen.

EXPERT TIPS**Pass**

In Python, if you're not sure what code you want inside a function yet, you can use the `pass` keyword in its place, and then come back to it later. It's a bit like skipping a question in a quiz but answering it later.

15 Time's up!

Now let's fill in the code for the `time_up()` function. In this function, set the `game_over` Boolean variable to `True`, which will tell the program to quit the game when the function is called. Type in the following code.

```
def time_up():
    global game_over
    game_over = True
```

Remember to delete the `pass` keyword and then add these lines.

16 Set the timer

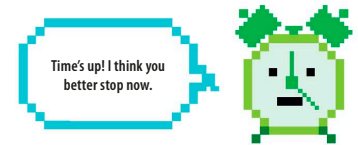
Now that `time_up()` is defined, the program needs to run it. But it needs to run seven seconds after the game starts. You can use Pygame Zero's built-in tool `clock` to do this. This tool lets the program call a function after a specified amount of time. Add this line in the code as shown here.

```
clock.schedule(time_up, 7.0)
place_coin()
```

This line will run the function `time_up()` seven seconds after the game starts.

17 Ending the game

The game starts and then seven seconds later, `clock.schedule` will run the `time_up()` function, which ends the game. But the game still needs to show the player's final score. For this, you need to add in one more bit of code to the `draw()` function.



```
def draw():
    screen.fill("green")
    fox.draw()
    coin.draw()
    screen.draw.text("Score: " + str(score), color="black", topleft=(10, 10))

    if game_over:
        screen.fill("pink")
        screen.draw.text("Final Score: " + str(score), topleft=(10, 10), fontsize=60)
```

If the variable `game_over` is `True`, this will turn the screen pink.

The final score is shown on the screen.

This command sets the size of the text shown on the screen.

18 Using update()

The final function you need to define is `update()`. This is a built-in Pygame Zero function, which means that unlike the other functions, you don't need to worry about when to run it. Once you've defined it, Pygame Zero will run it automatically—60 times a second! Delete `pass` under `def update()` and add this code. It will move the fox to the left if the left keyboard arrow is pressed.

```
def update():
    if keyboard.left:
        fox.x = fox.x - 2
```

This moves the fox two pixels to the left if the left arrow is pressed.

19 One way only

Now test your code. You should be able to move the fox to the left. But the fox needs to be able to move in other directions, too, so add this code to do that.



```
def update():
    if keyboard.left:
        fox.x = fox.x - 2
    elif keyboard.right:
        fox.x = fox.x + 2
    elif keyboard.up:
        fox.y = fox.y - 2
    elif keyboard.down:
        fox.y = fox.y + 2
```

The else-if branches are used to move the fox depending on which arrow key is pressed.

20 Collect the coins

Finally, you need to add some code that will update the score if the fox touches a coin. Add this code to the `update()` function.

Make sure you add this line at the very top.

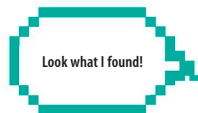
If the fox touches the coin, this variable will be True.

```
def update():
    global score
    if keyboard.left:
        fox.x = fox.x - 2
    elif keyboard.right:
        fox.x = fox.x + 2
    elif keyboard.up:
        fox.y = fox.y - 2
    elif keyboard.down:
        fox.y = fox.y + 2
    coin_collected = fox.colliderect(coin)
    if coin_collected:
        score = score + 10
        place_coin()
    clock.schedule(time_up, 7.0)
    place_coin()
```

This will increase the score by ten.

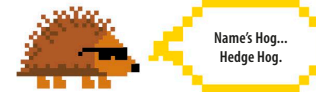
21 Game complete!

You've written all the code, and your game is now ready to go! Test your game and see how many coins you can collect before the game is over.



Hacks and tweaks

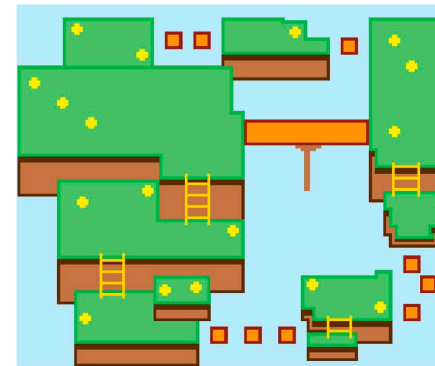
There are lots of ways to modify your game. You could try changing the fox to a different character of your choice, or you could make the game last longer.



```
hedgehog = Actor("hedgehog")
```

△ A different Actor

You can replace the fox with some other character by using another image from the Python Games Resource Pack, or you can use the 8-bit editors available online to make your own Actor. Remember to update the code so it uses the name of the new character throughout the program.



△ Change the playing area

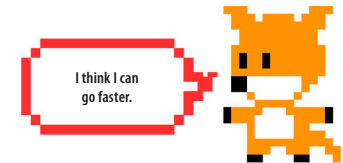
You can change the size of the playing area by changing the values of `WIDTH` and `HEIGHT`. Try using different numbers for these values and see what happens. Can you spot which part of the code you need to update?



```
clock.schedule(time_up, 15.0)
```

△ Extra time

The game currently ends after seven seconds. To make the game easier, you could give the player more time to play. You can do this by changing just one line of code.



```
if keyboard.left:
    fox.x = fox.x - 4
elif keyboard.right:
    fox.x = fox.x + 4
elif keyboard.up:
    fox.y = fox.y - 4
elif keyboard.down:
    fox.y = fox.y + 4
```

△ Go faster!

You can tweak the code to make the fox move faster. For this, you'll need to change some of the code in the `update()` function. At the moment, the fox moves two pixels every time the arrows are pressed. Here's a way to make it move at double that speed.