

Fruit Ninja Games

Introduction

We are going to build a few versions of the fruit ninja game. We will start with an easy version where every step (line of code) is given to you, but as you progress you will need to figure out more and more details yourself. At each stage you should make sure you understand what is happening, and if not then please ask.

① Basic fruit ninja game

Our starting version of the game consists of an apple that appears at random positions on the screen and the computer prints out "Good shot!" and draws a new apple when you click on the apple, otherwise prints "You missed!" and ends the game.

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

② Code review

The *Coding Games with Python* is very good, but it does do a few things that we can improve on. So before we move on we are going to look at our code and see if we can improve it.

③ Adding sounds

One of the reasons we switched over from turtle graphics to **Pygame Zero** is to simplify generating sounds — both short terms sound effects and long term background music.

④ Keeping score

Currently the game just prints out a different message whenever you miss or hit a fruit. It would be nicer to keep score so that you can show off your shooting skills. To do this we will create a new variable called score. Also we will add a heads up display (HUD) to show the score and game title.

⑤ Moving targets

Hitting a stationary object is easy ... what happens if they are moving?

So we are going to change our fixed position fruit to fruit that is thrown upwards from below the screen and then fall back down due to gravity — now we will need some physics!

⑥ Varying targets

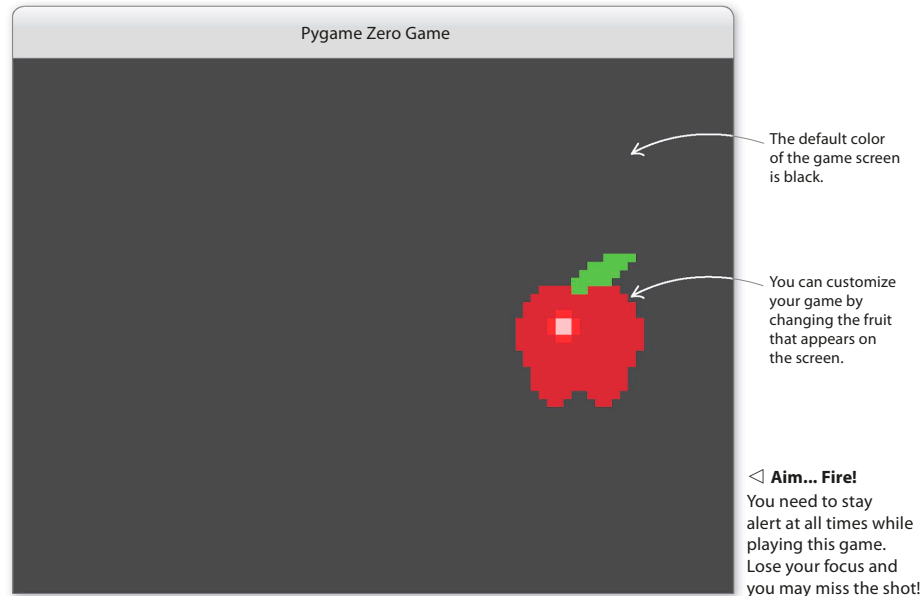
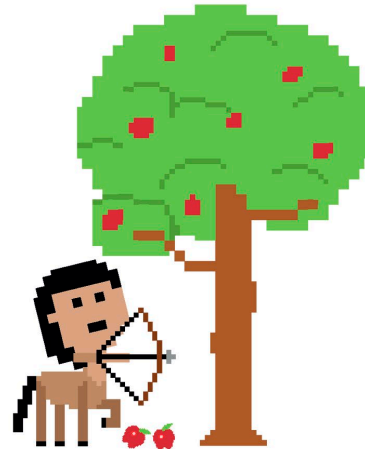
We don't just want apples! Lets add other fruit also, after all this is 'fruit ninja' not 'apple ninja'.

How to build Shoot the Fruit

This simple shooting game is a fun way to practice your aim. When the apple appears, click on it to "shoot" it. Aim carefully though, because if you miss, the game is over!

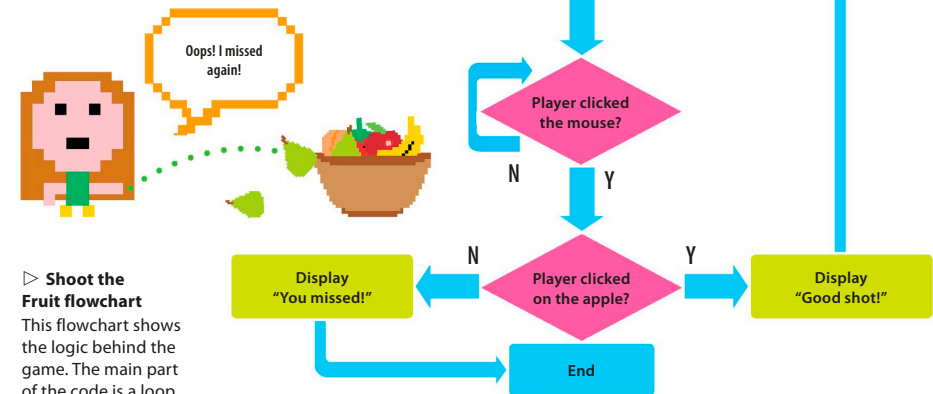
What happens

When the game starts, an apple appears on the screen for you to "shoot." If you hit it, a "Good shot!" message pops up, and the apple appears at another point on the screen. But if you miss, a "You missed!" message is shown, and the game ends.



How it works

The game is constantly checking whether you've clicked the mouse button. Every time you click on the apple, it needs to be drawn again somewhere else on the screen. If you click and miss, the game will end.



Shoot the Fruit flowchart

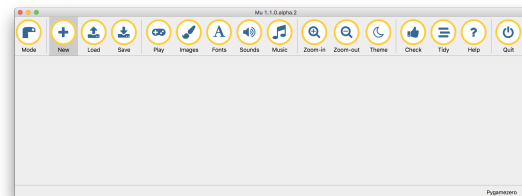
This flowchart shows the logic behind the game. The main part of the code is a loop that checks if you have clicked on the apple or not.

Get shooting!


Are you ready to code? In this program, you'll start by drawing an apple on the screen, then you'll learn to place it at random points before you start shooting it. Ready? Let's get coding!

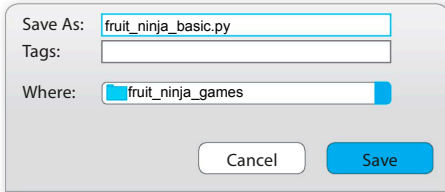
1 New File

In the mu-editor, click on the new file button,

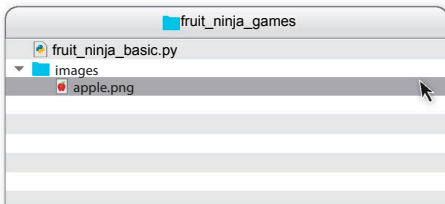


2 Save your game

To save your file click on the save button . Now save your program in the coderdojo_tramore folder. Create this folder now if you haven't made it already. Inside this folder, make another folder called fruit_ninja_games and save your file as fruit_ninja_basic.py

**4 Put the image into the folder**


Go to dk.com/computercoding and download the Python Games Resource Pack or just the Shoot the Fruit images. Find the file called "apple.png". Copy this file into the images folder. Your folders should look something like this now.



I wish my folders were more organized.

**3 Set up an image folder**

This game uses an image of an apple. Within your fruit_ninja_games folder you want to create a folder called images.

Click on the image button  and you should see an empty folder where you need to put your images.

I have put a copy of this resource pack in the folder resources

5 Introducing an Actor

Now you can start writing some code. Go back to Mu and write this line of code in the editor window, then press **Enter**.

```
apple = Actor("apple")
```

This creates a new Actor called apple.

**LINGO****Actors and Sprites**

In computer games development, a sprite is an object, like a coin or an enemy character, that is controlled by code. Actors in Python are like Sprites in Scratch. An Actor can be drawn on the screen, moved around, and even interact with other Actors in the game. Each Actor is given a "script" (the Python code) to tell it how to behave in the game.

6 Drawing the apple on the screen

Next you need to "draw" the apple on the screen. To do this, you can use a built-in Pygame Zero function called `draw()`. This function is used to redraw the game screen. For example, if a character has moved or a score has changed, you can use this function to update the screen with those changes. Write this code beneath your previous code.

```
apple = Actor("apple")

def draw():
    screen.clear()
    apple.draw()
```



This function is called `draw()`.

This clears the screen.

Remember, you need four blank spaces here.

This line draws the apple on the screen.

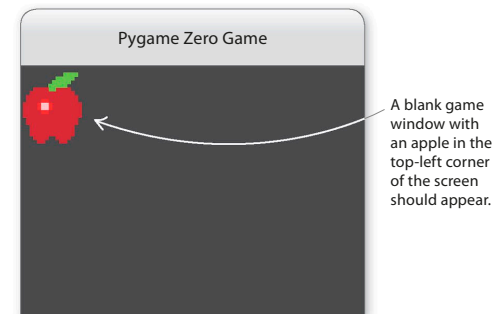
7 Test your code

To test your code, click on the play button . To stop, click on the stop button, .

If you need help running your game, check pages 24–25.

**8 First screen**

If your code is working properly, you should see this screen. If it's not, or if you get an error message, go back and check your code to find any bugs.

**9 Placing the apple**

At the moment, the apple appears in the top-left corner of the game window. You can change the code to place the apple exactly where you want it on the screen. Write this function, which will place the apple at the coordinates (300, 200).

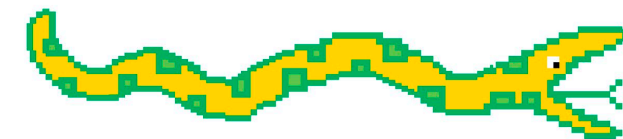
```
apple.draw()

def place_apple():
    apple.x = 300
    apple.y = 200
```

The apple will be placed 300 pixels along the x-axis (horizontal).

The apple will be placed 200 pixels down the y-axis (vertical).

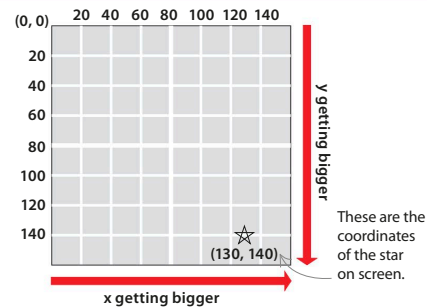
You must save your program before running it, or I'll run an old version of your code.



EXPERT TIPS

Graphics in Pygame

Python uses coordinates to identify all the places in a window where an object could be. This means that every place can be identified by using two numbers. The first number is the x coordinate, which shows how far to the right an object is. The second number is the y coordinate, which shows how far down the object is. Coordinates are written in parentheses, with the x coordinate first, like this: (x, y). In math, the coordinate (0, 0) is usually at the bottom left, but in computer graphics, it's almost always the top left.

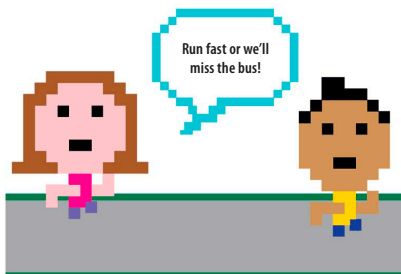


10 Running the function

After you've written the function to place the apple on the screen, you need to tell Python to run it. Add this extra line of code to run the function called `place_apple()`.

```
def place_apple():
    apple.x = 300
    apple.y = 200
    place_apple()
```

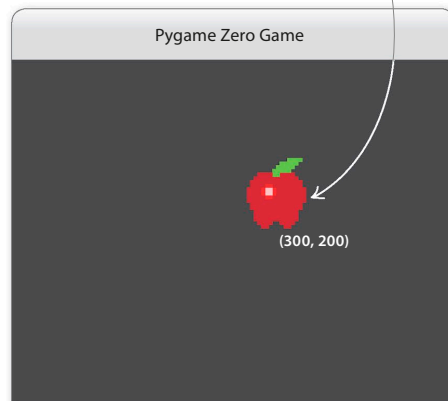
This function places the apple at coordinates (300, 200).



11 Test it again

Save your file and then run the code from the command line. Remember, you can press the **Up** arrow in the command line to quickly choose a previous command, then press **Enter**. This time the apple will appear at the point (300, 200).

The apple is placed at coordinates (300, 200).



12 Dealing with clicks

Now it's time to write the code that will run when you press the mouse. Pygame Zero has a built-in function called `on_mouse_down()`, which is run every time you click the mouse. Type this code in between the code you added in Step 9 and Step 10, then run it from the command line. You should see the message "Good shot!" in the Command Prompt or Terminal window each time you click the mouse.

```
def place_apple():
    apple.x = 300
    apple.y = 200
    def on_mouse_down(pos):
        print("Good shot!")
        place_apple()
    place_apple()
```

Programmers sometimes add blank lines to make their code neater, but they aren't necessary. Python ignores blank lines completely.

13 Adding some logic

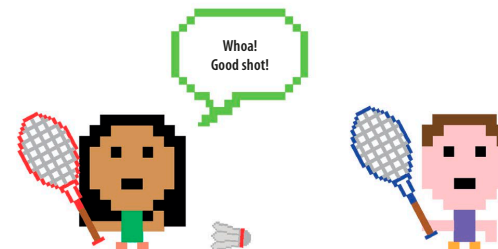
At this point, the "Good shot!" message is displayed every time you click the mouse, but we only want it to show if the player actually hits the apple. You can do this by amending the code from Steps 10 and 12 to include an if statement. This code checks if the apple and the mouse cursor are in the same position. If they are, the message is displayed.

```
def on_mouse_down(pos):
    if apple.collidepoint(pos):
        print("Good shot!")
        place_apple()
```

`pos` is the position of the cursor when you click the mouse.

Make sure the bottom two lines now start with eight spaces.

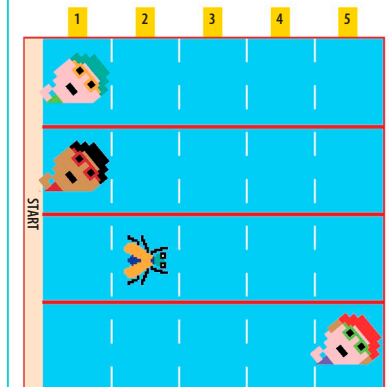
This function checks if the cursor is in the same position as the apple.



EXPERT TIPS

Indents

Python uses indents to separate different blocks of code. If your indents are wrong, Python won't know how to read your code, and you'll end up with a bug! Each indent is made up of four spaces, and code can be indented more than once—for example, by eight spaces. Sometimes IDLE adds the indents for you, but if you're moving the code around, like in Step 13, you might need to indent it yourself. You can do this by entering the correct number of spaces.





56

SHOOT THE FRUIT

14

Missed a shot? Game over!

Add some more logic to your code, so that if you miss a shot and don't click on the apple, it quits the game. Try it out!

```
if apple.collidepoint(pos):
    print("Good shot!")
    place_apple()
else:
    print("You missed!")
    quit()
```

This command quits the game by stopping the program completely.

16

Using Random

Change the code you typed in Step 9 to look like this. The code will now use the **randint()** function to pick a random number between 10 and 800 for the x coordinate and a random number between 10 and 600 for the y coordinate.

```
def place_apple():
    apple.x = randint(10, 800)
    apple.y = randint(10, 600)
```

This function picks a random number for each coordinate.

15

Importing Random

The game is very easy at this point, because the apple is always drawn at the same place on the screen. You can use Python's Random module to make the game more challenging by placing the apple at a random point on the screen each time it is drawn. First, add this code at the very top of your program.

```
from random import randint
apple = Actor("apple")
```

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

17

Time to shoot!

You did it! Run your program to play the game. Each time you "shoot" the apple, it will move to a random place on the screen for you to "shoot" again.

**EXPERT TIPS****Random numbers**

Rolling a dice, picking a card from a deck, or tossing a coin are all actions that you can simulate by generating a random number. You can read more about how to use Python's Random module by going to the **Help** menu and clicking **Python Docs**.






OK, that is our basic game idea, we will now start to add more and more complicated features to it.



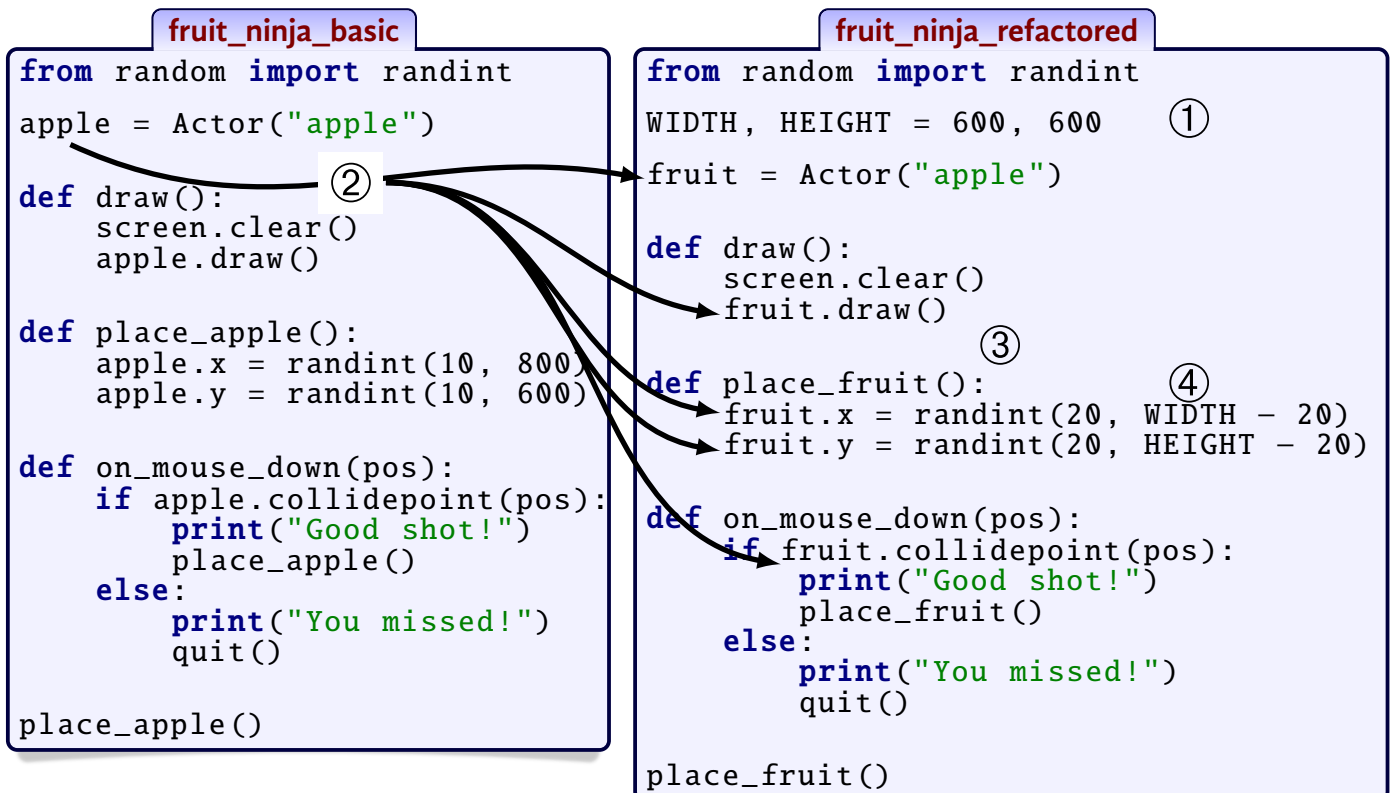
1 Code review

The *Coding Games with Python* is very good, but it does do somethings that we can improve on. Before we move on we are going to look at our code and see if we can improve it. We are not trying to change what it does — that will come later — instead we want to make the code easier for humans to read and easier for humans to update when they think of something new to add to the game.

So we will make a number of changes:

 Click on new file icon, , and copy the code from `fruit_ninja_basic` into this new file. Save, using icon , as file `fruit_ninja_refactored.py`. Finally update the file using the following steps:

- ① We set the size of the screen using `WIDTH` and `HEIGHT` global variables.
- ② Change name from `apple` to `fruit` as later we want to deal with more than apples.
- ③ Change name of function `place_apple` to `place_fruit` as we will use this function to place all fruits.
- ④ Change the limits in function `randint` so that the fruit will appear fully on the screen. Now, because we have define global variables `WIDTH` and `HEIGHT` we can limit the random position of the fruit based on the screen size.



What is code refactoring?

Code refactoring is the process of restructuring existing computer code to improve its readability or structure, with the explicit purpose of keeping its meaning or behaviour. Refactoring is the part of code maintenance which doesn't fix bugs or add new functionality. Rather it is designed to improve the understandability of the code, to make it easier for human maintenance in the future.



2 Adding sound effects

One of the reasons we switched over from turtle graphics to **Pygame Zero** is to simplify generating sounds — both short term sound effects and long term background music.

Pygame Zero treats sound effects and background music separately.

2.1 (Short term) Sound effects

A sound effect is a short (less than a few seconds) audio that is usually played in response to something happening in a game — target was hit, player died, etc.. To create sound effects you (I have carried out these steps already for you, but list them here so you can try out your own sound effects):

- Find a suitable sound effect. The website www.zapsplat.com, has an excellent search engine and a large collection of free sounds. You need to pay for the WAV format, but the MP3 format is free. I always use the MP3 format.



For example, for this game I searched “target” and found sound ‘Clay pigeon shooting, gun shoots and people cheer, UK 2’. This has a splay sound and a cheer — I will use both.

- Convert the sound to WAV format. I use the free audio editor Audacity, (www.audacityteam.org), to edit sound effects and exporting them as MP3 format.



For example, using the Clay pigeon shooting, gun shoots and people cheer, UK 2 sound I saved two WAV file — one contained just the splat sound and the second the splat and cheer sound.



- Copy the sound file into the sounds sub-folder of `fruit_ninja_games`. Make sure that you rename your file so that it uses lower case letters, underscore and numbers only, and that it starts with a letter.

For example, in this game I have two sound effects called `shoot.wav` and `shoot_and_cheer.wav`

- Now wherever in your code you want to play sound, `NAME`, you use `sounds.NAME.play()`.

For example, I wanted to play the ‘shoot’ sound every time the target is hit and on average every ten hits it plays the ‘shoot_and_cheer’ also so I wrote the following code



Click on new file icon, , and copy the code from `fruit_ninja_refactored` into this new file. Save, using icon , as file `fruit_ninja_sounds.py`. Finally, insert the following code.

```

19  if fruit.collidepoint(pos):
20      print("Good shot!")
21      if randint(1,10)==1:
22          sounds.shoot_and_cheer.play()
23      else:
24          sounds.shoot.play()
25      place_fruit()
```



2.2 (Long term) Background music

Background music is usually a longer piece of audio and often run in a loop so it repeats until the game is over. To play background music you:

- Find a suitable sound track. I like the website www.melody-loops.com. It has a good selection of tracks and it has a nice feature of cutting the tracks to whatever length you want and control fade in at the start and fade out at the end of the tracks.



For example, for this game I picked The French Lovers Song and set its length to be 90 seconds with a few second fade out at the end.

- Copy the sound file into the `music` sub-folder of `fruit_ninja_games`. Make sure that you rename your file so that it uses lower case letters, underscore and numbers only, and that it starts with a letter.

I renamed the downloaded file as `the_french_lovers_song.mp3`.

- Then to
 - play this track in a loop forever, use code: `music.play("the_french_lovers_song")`.
 - play this track once, use code: `music.play_once("the_french_lovers_song")`.
 - stop the currently playing track, use code: `music.stop()`.
 - pause use code: `music.pause()` and to unpause, use code: `music.unpause()`.
 - fade out and stop after 'duration' seconds, use code: `music.fadeout(duration)`.
 - set volume with value in range 0 ... 1, use code: `music.set_volume(volume)`.





Add code to play the `the_french_lovers_song` or to play `trap_ashamaluev_music` which I found on soundcloud and also downloaded to `music` sub-folder.

3 Keeping score (and multiple lives)

3.1 Keeping score

Currently the game just prints out a different message whenever you miss or hit a fruit. It would be nicer to keep score so that you can show off your shooting skills. To do this we will create a new variable called `score`. Also we will add a heads up display (HUD) to show the score and game title.



Click on new file icon, , and copy the code from `fruit_ninja_sounds` into this new file. Save, using icon , as file `fruit_ninja_score.py`. Finally, update the code as follows.

Step 1) Create variables to store the required information.

I like to store this information within an actor. So I after creating the actor `fruit` I write the following:

```
fruit.score = 0
```

**Step 2) Update variables when needed.**

For the score, we want to increase this by one every time we hit the target, so we have

```
if fruit.collidepoint(pos):  
    fruit.score = fruit.score + 1
```

Step 3) Display information.

For all our games, I'm going to display the score on the top left, the game name in the centre, and lives (or other information) on the top right. So the screen looks like this.



The code to produce the game title and score is as follows:

```
screen.draw.text("Score %s" % fruit.score,  
    topleft=(0, 0))  
screen.draw.text("Fruit Ninja",  
    midtop=(WIDTH // 2, 0), color="orange", fontsize=60)
```

3.2 Multiple lives



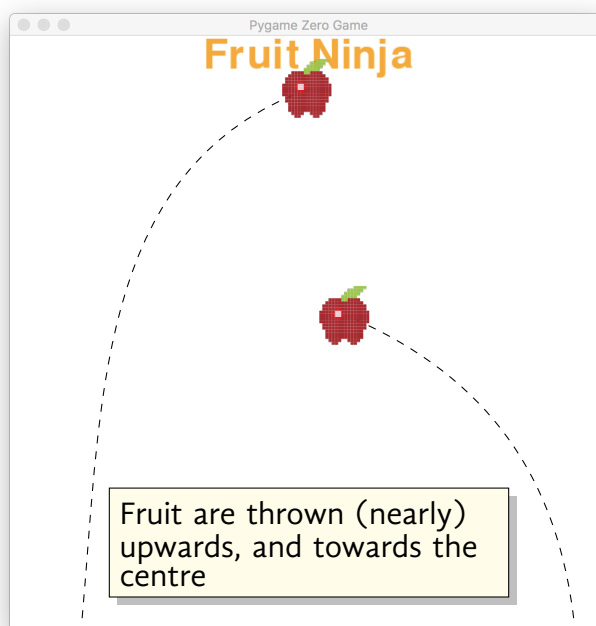
Now you need to write similar code for multiple lives. You start with three lives and every time you miss a target you lose a life. When you have no lives left the game is over.

4 Moving targets

Rather than placing the fruit in a random position we could make the game much more interesting by having moving fruit. I think that the nicest effect is to 'throw' fruit up from below the screen and have them fall back down due to gravity. This sounds complicated but it's actually easy to do.



Click on new file icon, , and copy the code from `fruit_ninja_sounds` into this new file. Save, using icon , as file `fruit_ninja_physics.py`. Finally, update the code as follows.



All fruit start below the screen

**Step 0) Plan you code.**

The function `place_fruit` does a few things so lets be good programmers and plan what we are going to do.



Import the 2-dimensional vector from `pygame.math` library at the start of your program so that we can treat our horizontal (x) and vertical (y) movement together.

```
from pygame.math import Vector2 as Vector
```



Change the function `place_fruit` so it just contains the following comments. We will fill in the details or each in the following sections.

```
def place_fruit():  
  
    # random position below the screen  
  
    # random velocity – fruit thrown towards the centre  
  
    # acceleration due to gravity
```

Step 1) Set the starting position.

We want the fruit to appear anywhere along the bottom edge of the screen — so we set the x -coordinate to be random value in the range 20 to `WIDTH-20`.

We want the fruit to appear from below the screen — so we set the y coordinate to be the screen height plus some margin.

```
# random position (just below the screen)  
x = randint(20, WIDTH - 20)  
y = HEIGHT + 20  
fruit.pos = Vector(x, y)
```

Step 2) Set the starting velocity.

We want the fruit to be thrown upward to a random height so dy , the velocity along the y -direction, is set to be a random value in range 10 to 20. These numbers were picked by trial and error, so the fruit would just go up to top of screen before falling back down.

```
# random velocity – fruit thrown towards the centre  
dy = -randint(20, 25)
```

Note the minus sign. This is because y -direction points downwards.

If the x -direction component the velocity, dx , is zero then the fruit will just go directly up and then fall back down — this is a bit boring. But if we set dx to a too large a value the fruit will fly past the side of the screen, and we won't have time to click on it. So we will change the sign of dx so that the fruit is thrown towards the centre.

```
dx = randint(0, 5)  
if fruit.x > WIDTH / 2:  
    dx = -dx
```



Finally, we now can set the velocity

```
fruit.velocity = Vector(dx, dy)
```

Step 3) Set the acceleration.

On Earth acceleration due to gravity is 9.81 metres per second squared, but in our game time does not run at the same rate so we can use a smaller value

```
# acceleration due to gravity  
fruit.acceleration = Vector(0, 0.5)
```

Step 4) Update velocity and position.

So now, all we have done is set the starting state of the fruit. How does it move?

Physics — or a sub-area of physics, called projectile movement — comes to our aid.

We have an object, an apple, that has position and velocity and is subjected to a fix acceleration¹ (gravity) how does it move. We have two equations

$$\begin{aligned}\text{velocity} &= \text{rate of change of position} \\ \text{acceleration} &= \text{rate of change of velocity}\end{aligned}$$

What is the 'rate of change' of something? It is the change (new – old) divided by the length of time over which the change occurred. So we have

$$(\text{velocity}) = (\text{rate of change of position}) = \frac{(\text{new position}) - (\text{old position})}{(\text{time duration})}$$

Rearranging this we get

$$(\text{new position}) = (\text{old position}) + (\text{time duration}) \times (\text{velocity})$$

In our game world we will let (time duration) to be equal to one so the above equation to update position can be written as

```
fruit.pos = fruit.pos + fruit.velocity
```

We do this type of updating so often that python has an alternative, more compact, style

```
fruit.pos += fruit.velocity
```

Using the same logic we can go from equation

$$\text{acceleration} = \text{rate of change of velocity}$$

to python code

```
fruit.velocity += fruit.acceleration
```

¹Remember, unlike in everyday speech, 'acceleration' does not just mean we go faster. It means that our velocity changes, i.e. speed increases or decreases or/and direction changes.



We now want to run the two lines to update velocity and position over and over and over and over and We do this by defining the function `update` when is a special function that runs 60 times a second.

 Insert the following function.

```
def update():
    # apply physics
    fruit.velocity = fruit.velocity + fruit.acceleration
    fruit.pos = fruit.pos + fruit.velocity

    # fell below screen so get new fruit
    if fruit.y > HEIGHT + 30:
        place_fruit()
```

Note:

- We always update velocity before we update position. This will give us more accurate behaviour to help correct the fact that we only update 60 times a second but in reality Physics is update continuously.
- When the fruit falls below the screen again, we ‘throw’ a new fruit.

Update: You can also get a rotating effect by changing the property `angle` of you fruit.

5 Multiple targets (and not all fruit!)

We have one final improvement to make to our game. Rather than always throwing an apple we could throw different fruit. Or even better sometimes throw non-fruit items which we are NOT to click on.

I won't write the steps to do this here — instead we will cover this in the dojo — however we need some images.

For game art, I like the site www.gameartguppy.com which was developed by Vicki Wenderlich to “give game devs who can't afford custom art (yet) an easy way to find and use free and inexpensive art.”

I have put in some of the food collection of images that I downloaded from www.gameartguppy.com into `coderdojo-tramore/resources` folder.

