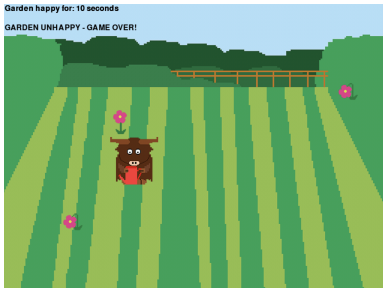
25

Gardening Games

Introduction

Gardening games are never ending games where you need to keep various object healthy (here watering flowers) within various constraints (here a time constraint and avoiding mutant flowers). Depending on the game parameters this can be either a relaxing distraction or an intense challenge.



① **Basic game** ..... `gardening_games_basic.py`

Our starting version of the game comes from the excellent *Coding Games with Python* book.

② **Adding sounds** ..... `gardening_games_sound.py`

Go to www.zapsplat.com, and find some sounds that we can use for the various game events, including:

- flower grows
- flower droops due to needing water
- mutant flower is generated due to too much watering.

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** ..... `gardening_games_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 ways we can tweet the starting game idea

CoderDojo, Tramore, Waterford. (`kmurphy@wit.ie`)

How to build Happy Garden

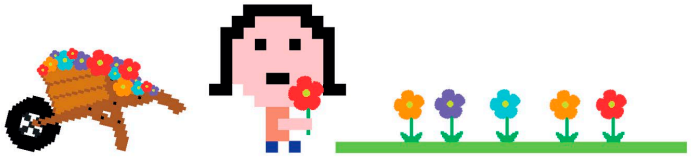
Gardening may seem like a relaxing hobby, but not in this game! Can you help a flower-loving cow keep all the plants watered? Look out for the scary fangflowers as they try to zap the cow. How long can you help it keep the garden happy?

What happens

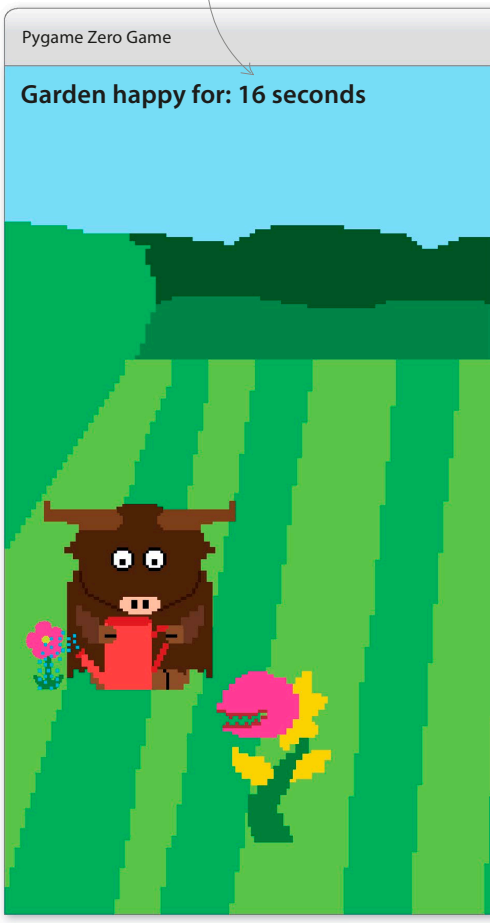
When the game starts, a cow with a watering can appears in the garden, but there is only one flower. Every few seconds another flower appears or an existing flower begins to wilt. Use the arrow keys to move the cow to the wilted flowers and press the **Space bar** to water them. If any flower remains wilted for more than ten seconds, the game ends. But if the garden is happy for more than 15 seconds, one of the flowers mutates into a fangflower and tries to zap the cow.

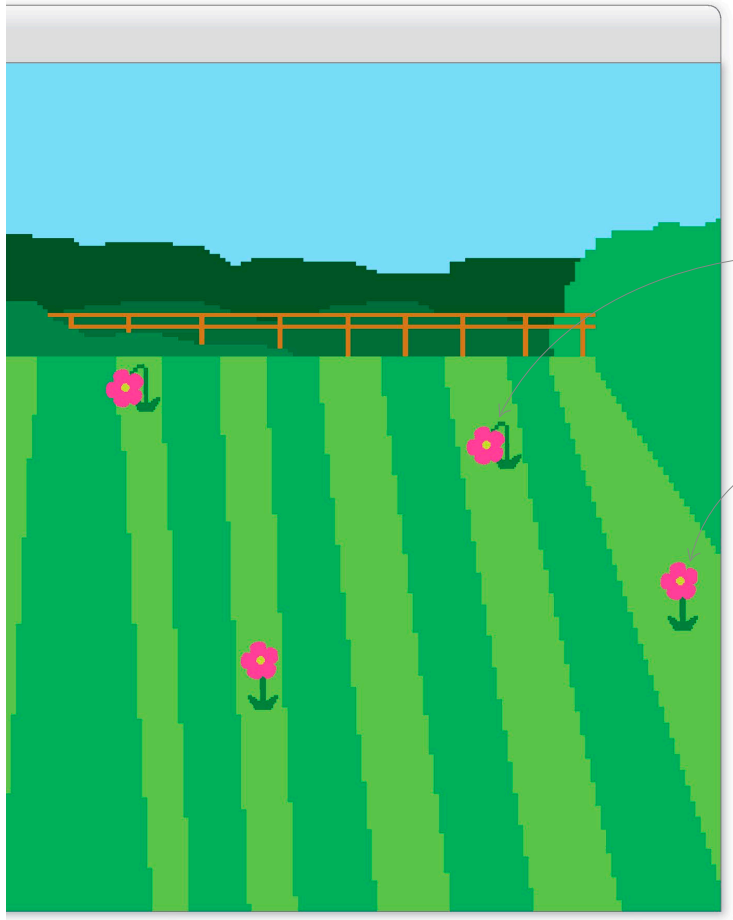
◁ **Cow**
The cow is the main character in the game. Its aim is to keep all the flowers watered.

◁ **Fangflower**
This large carnivorous plant moves around the garden and tries to zap the cow.



The counter displays the number of seconds the garden has been happy for.





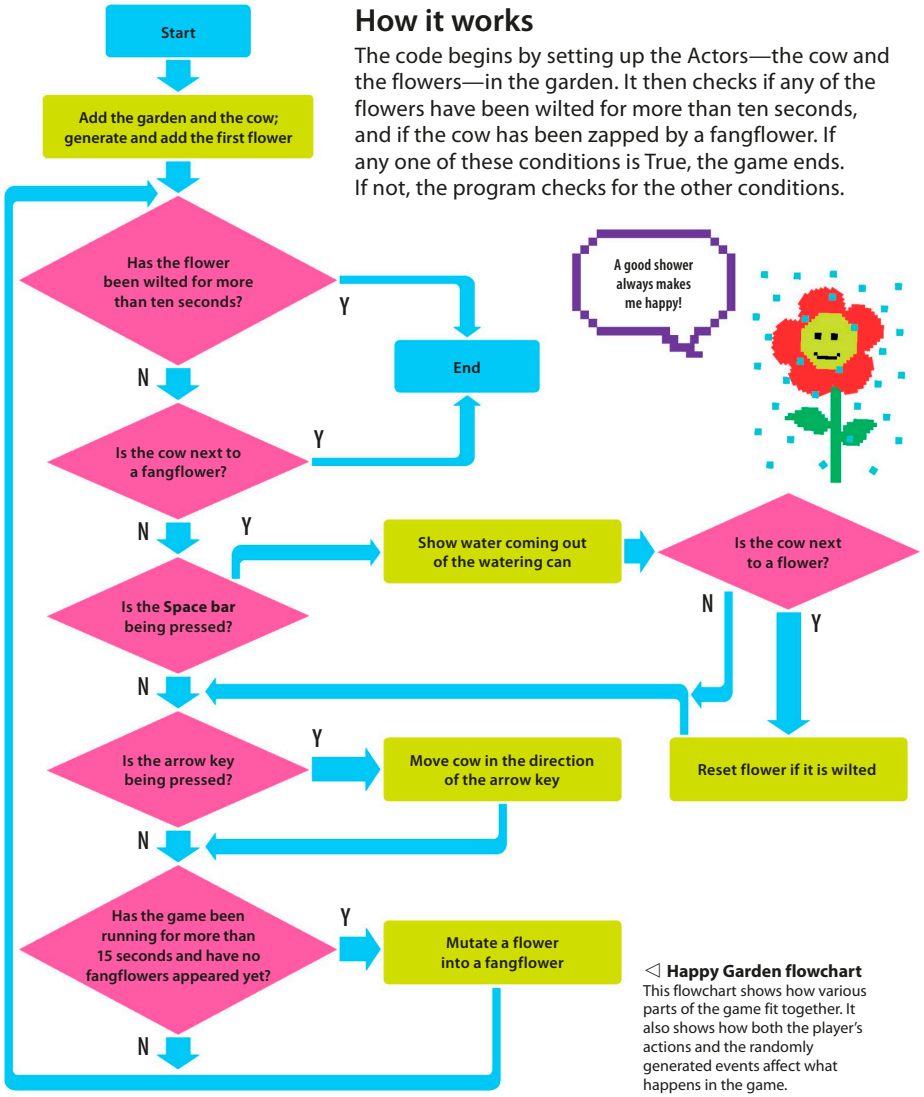
If a flower remains wilted for more than ten seconds, the game ends.

The blooming flowers continue to appear at random positions on the screen as the game progresses.

◁ **Keep moo-ving!**
There are a lot of different elements in this game. The code uses several functions to keep track of them all.

How it works

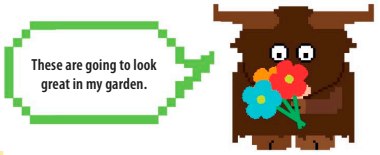
The code begins by setting up the Actors—the cow and the flowers—in the garden. It then checks if any of the flowers have been wilted for more than ten seconds, and if the cow has been zapped by a fangflower. If any one of these conditions is True, the game ends. If not, the program checks for the other conditions.



◁ **Happy Garden flowchart**
This flowchart shows how various parts of the game fit together. It also shows how both the player's actions and the randomly generated events affect what happens in the game.

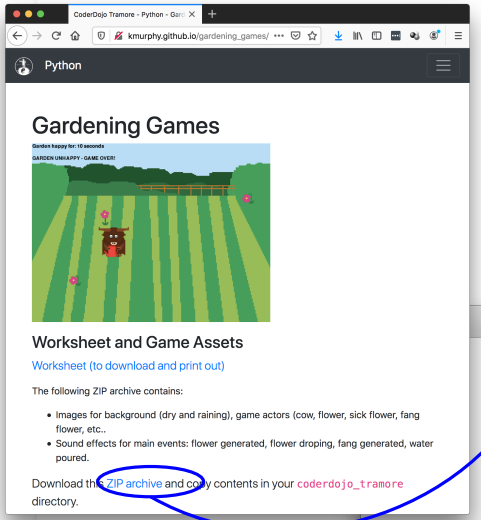
It's gardening time!

It will take some preparation to get the garden ready. Begin by setting up the folders and downloading the images you'll need.



1 Game Setup

Go to `kmurphy.github.io/gardening_games`



2

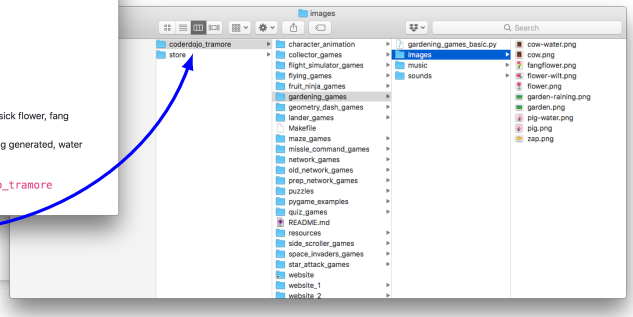
Download the ZIP archive of game assets.

3

Extract archive contents to `coderdojo_tramore` directory. (This will create a new directory called gardening_games.)

4

In `Mu` editor create a new file and save it in `gardening_games` using name `gardening_games_basic.py`.



5 Import modules

It's time to start coding. Go back to your garden.py file and start by importing some modules. You'll use **randint()** to randomly choose which flowers will wilt or mutate. The functions in the Time module will keep track of how long the garden has been happy for or how long any flowers have been wilted.

```
from random import randint
import time
```

This imports Python's Time module.

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

6 Declare global variables

Next define the global variables. These are the variables that will be used throughout the game. Type this code under the lines you added in Step 5.

```
WIDTH = 800
HEIGHT = 600
CENTER_X = WIDTH / 2
CENTER_Y = HEIGHT / 2

game_over = False
finalized = False
garden_happy = True
fangflower_collision = False

time_elapsed = 0
start_time = time.time()
```

These variables define the size of the game screen.

These are flag variables, which let you know what's happening in the game.

These variables help keep track of the time.

7 Add the cow

To start with, the only Actor in the game is the cow. Type the code shown in black to add the cow and set its starting position.

```
start_time = time.time()

cow = Actor("cow")
cow.pos = 100, 500
```

These values set the cow's starting position on the screen.

8 Create lists for other Actors

The other Actors in the game—flowers and fangflowers—are generated at random as the game progresses. Since you don't know how many of them will be generated, create lists to store each one that appears.

```
flower_list = []
wilted_list = []
fangflower_list = []
```

Each time a new flower Actor is created, it gets added to this list.

This list will store how long a flower has been wilted.

This list will store the fangflower Actors.

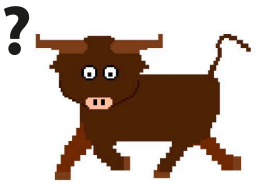
9 Keep track of the fangflowers

In this game, you need to pay special attention to the zapping fangflowers. You'll need to make them move around the garden. They also need to bounce off the edges of the game window so that they are always on the screen. You can do this by keeping track of their velocity—the speed at which something moves in a particular direction—along the x-axis and the y-axis. Add these lines after the code from Step 8.

```
fangflower_vy_list = []
fangflower_vx_list = []
```

This will hold the velocities of the fangflowers along the y-axis.

This will hold the velocities of the fangflowers along the x-axis.



10 Draw the garden

Now that you've set up those variables, you need to draw the garden and the cow. There are no flowers or fangflowers to draw yet, but add the code that will draw them when they're generated. Add these lines immediately after the code from Step 9.

```
def draw():
    global game_over, time_elapsed, finalized
    if not game_over:
        screen.clear()
        screen.blit("garden", (0, 0))
        cow.draw()
        for flower in flower_list:
            flower.draw()
        for fangflower in fangflower_list:
            fangflower.draw()
        time_elapsed = int(time.time() - start_time)
        screen.draw.text(
            "Garden happy for: " +
            str(time_elapsed) + " seconds",
            topleft=(10, 10), color="black"
        )
```

This code draws the cow on the screen.

This code will draw all the flowers.

This will draw all the fangflowers.

This code checks how long the game has been running for.

11 Time to test

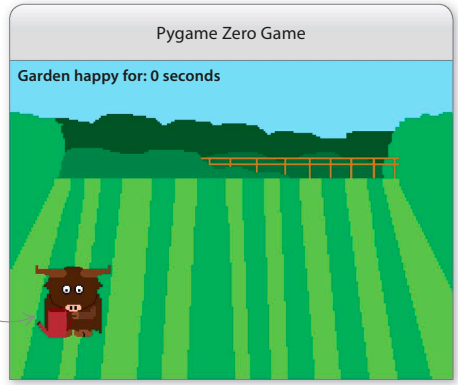
It's time to take a look at your garden! Save the IDLE file and then run it from the command line in the Command Prompt or Terminal window.

```
pgzrun
```

Type this in the Command Prompt or Terminal window and then drag and drop the garden.py file.

12 Get a sneak peek!

If there are no mistakes in your code, you should see a screen like this. If something's wrong, don't worry! Just use your debugging skills to check that you've spelled everything correctly and used the correct number of spaces for indentation.



You should see the garden and the cow holding a watering can.

13 Other functions

You'll use a lot of functions in this game. You can list some of them now and define them later in the code. Using **pass** will make sure that Python doesn't run anything yet. Type this code under what you added in Step 10.

```
def new_flower():
    pass

def add_flowers():
    pass

def check_wilt_times():
    pass

def wilt_flower():
    pass

def check_flower_collision():
    pass

def reset_cow():
    pass

def update():
    pass
```

Don't forget to save your work.

14 Moving around the garden

At the moment, the cow appears on the screen but doesn't do anything. Add some code that lets you move the cow around. Replace the word **pass** under **def update()** with the following code.

This moves the cow five pixels to the right when the **Right** arrow key is pressed.

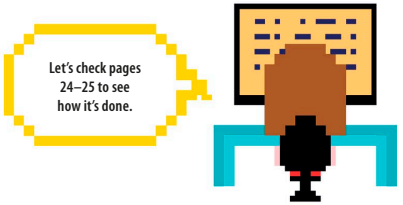
```
def update():
    global score, game_over, fangflower_collision
    global flower_list, fangflower_list, time_elapsed
    if not game_over:
        if keyboard.left and cow.x > 0:
            cow.x -= 5
        elif keyboard.right and cow.x < WIDTH:
            cow.x += 5
        elif keyboard.up and cow.y > 150:
            cow.y -= 5
        elif keyboard.down and cow.y < HEIGHT:
            cow.y += 5
```


15 Another test

Test your newly updated code to make sure it's correct. Save your IDLE file and run it from the command line. You should now be able to move the cow on the screen.

```
pgzrun
```

Type this in the Command Prompt or Terminal window and then drag and drop the garden.py file.



16 Add a flower

In this step, you'll create a flower Actor for the cow to water and add it to the end of **flower_list**. You'll also add the value **happy** to the end of **wilted_list**, which holds the amount of time each flower has been wilted for. The **happy** value will let the program know that the flower hasn't wilted. Replace **pass** in the **new_flower()** function with this code.

These are the global variables this function uses.

This line creates a new flower Actor.

This line sets the position of the new flower.

```
def new_flower():
    global flower_list, wilted_list
    flower_new = Actor("flower")
    flower_new.pos = randint(50, WIDTH - 50), randint(150, HEIGHT - 100)
    flower_list.append(flower_new)
    wilted_list.append("happy")
    return
```

This lets the program know that the flower is not wilted.

This adds the new flower to the list of flowers.

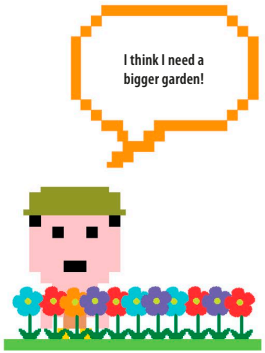
17 Add more flowers to the garden

Having just one flower to water would make the game too easy. You need to add some code that will create a new flower every four seconds to keep the cow busy. Add this code to replace the word **pass** under **def add_flowers()**.

This line calls the **new_flower()** function to create a new flower.

This adds a new flower every four seconds.

```
def add_flowers():
    global game_over
    if not game_over:
        new_flower()
        clock.schedule(add_flowers, 4)
    return
```



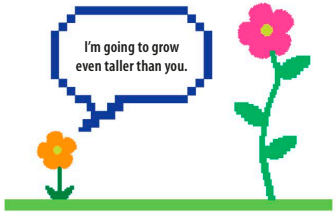
18 Start adding flowers

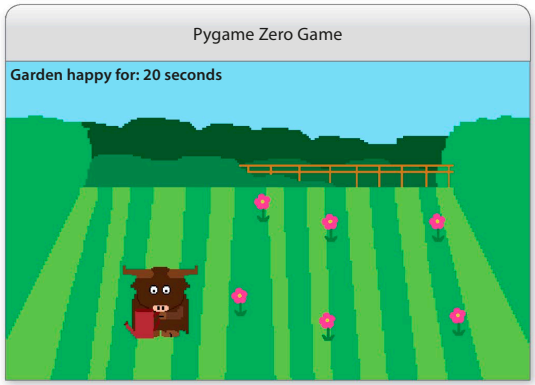
Although **add_flowers()** will schedule a call to itself every four seconds, you need to call it once in the program to start this process. Add the line in black above **def update()** from Step 13. Save and run your code to check if the flowers start appearing!

```
def reset_cow():
    pass

add_flowers()

def update():
```

19 Blooming garden

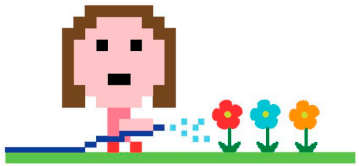
If there are no errors in your code, you will see a new flower appear on the screen every four seconds. This is what your screen will look like after 20 seconds. Use the arrow keys to move the cow around.





20 Water the flowers

It's time for the cow to water the wilted flowers. Let's add some code that will make the cow sprinkle water on the flowers when the player presses the **Space bar**. The code will also check if the cow is standing next to a flower. Add this code to the **update()** function.



```
    global flower_list, fangflower_list, time_elapsed
    if not game_over:
        if keyboard.space:
            cow.image = "cow-water"
            clock.schedule(reset_cow, 0.5)
            check_flower_collision()
        if keyboard.left and cow.x > 0:
            cow.x -= 5
```

This will check if the **Space bar** is being pressed.

This will change the image of the cow to the one with water coming out of the watering can.

This will reset the cow's image after half a second.

This line will check if the cow is next to a flower.

21 Stop watering

The code from Step 20 uses two functions that you haven't written yet—**reset_cow()** and **check_flower_collision()**. Let's add the code that will change the image of the cow using the watering can back to the version where it's just holding it. Replace the word **pass** under the **reset_cow()** function from Step 13 with the code in black below.

```
def reset_cow():
    global game_over
    if not game_over:
        cow.image = "cow"
    return

add_flowers()
```

This code runs if the game is not over yet.

This changes the cow's image back to the original one.

22 Points for accuracy!

You need the code to check if the cow is near a flower when the **Space bar** is pressed. If it is, the flower will get watered and its "time since flower wilted" value in **wilted_list** will be set to **happy**. To do this, you'll use Pygame Zero's built-in **collideret()** function to check if a flower and the cow have collided or are next to each other. Replace **pass** under **def check_flower_collision()** in Step 13 with the code shown below.

```
def check_flower_collision():
    global cow, flower_list, wilted_list
    index = 0
    for flower in flower_list:
        if (flower.colliderect(cow) and
                flower.image == "flower-wilt"):
            flower.image = "flower"
            wilted_list[index] = "happy"
            break
        index = index + 1
    return
```

These are the global variables you will use in this function.

This code loops through all the flowers in the list.

This variable helps the program to move through the list in order.

This condition applies if the cow is next to the flower you're looking at.

This changes the wilted flower's image back to the original version.

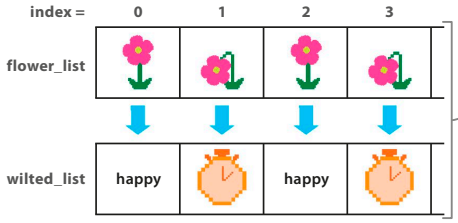
This line updates the value of **index** so that the program moves through the list.

This stops the loop from checking the other flowers.

This line stops the program counting how long the flower's been wilted.

23 Wilt a flower

It's time to give your garden-tending cow a bit of a challenge. Add some code that will wilt a random flower every three seconds. The cow will have to dash to the wilted flower to water it. Replace the word **pass** in the **wilt_flower()** function from Step 13 with the code below.



These two lists store the information of a particular flower at the same index number.

```
def wilt_flower():
    global flower_list, wilted_list, game_over
    if not game_over:
        if flower_list:
            rand_flower = randint(0, len(flower_list) - 1)
            if (flower_list[rand_flower].image == "flower"):
                flower_list[rand_flower].image = "flower-wilt"
            wilted_list[rand_flower] = time.time()
        clock.schedule(wilt_flower, 3)
    return
```

This line resets the time for this flower in **wilted_list** to the current time.

This line generates a random index in the list of flowers.

This checks if the flower at this index is wilted or not.

This schedules another call to **wilt_flower()** in three seconds.

This sets the flower image to a wilted flower image.

24 Unhappy garden!

Next you need to check if any of the flowers have been wilted for more than ten seconds. If one has, the garden's unhappy, and it's game over! Go to **def check_wilt_times()** in Step 13 and replace **pass** with the code shown here.

This code loops over each item in the **wilted_list**.

```
def check_wilt_times():
    global wilted_list, game_over, garden_happy
    if wilted_list:
        for wilted_since in wilted_list:
            if (not wilted_since == "happy"):
                time_wilted = int(time.time() - wilted_since)
                if (time_wilted) > 10.0:
                    garden_happy = False
                    game_over = True
                    break
    return
```

This checks if there are any items in the **wilted_list**.

These lines check if the flower is wilted and work out how long it's been wilted.

This line checks if the flower has been wilted for more than ten seconds.

25 Start wilting

Now that you've added a function to wilt the flowers, you need to call it. Add this code just below the call to **add_flowers()** that you added in Step 18.

```
        cow.image="cow"
    return

add_flowers()
wilt_flower()
```

This command will make the flowers wilt.

26 Check for happiness

Now you need to add a call to the **check_wilt_times()** function you defined in Step 24. Go to the **update()** function and add this line.

```
def update():
    global score, game_over, fangflower_collision
    global flower_list, fangflower_list, time_elapsed
    check_wilt_times()
    if not game_over:
```

This checks how long the flowers have been wilted.

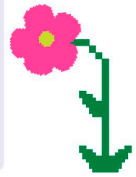
27 Game over!

Your game is almost ready! But before testing it, you need to add some code that lets the player know that the game is over if the flowers have been wilted for too long. Add an **else** branch to the **if not game_over** statement in the **draw()** function you defined in Step 10.

This line displays a message to show how long the garden has been happy.

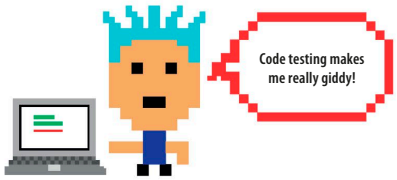
```
            str(time_elapsed) + " seconds",
            topleft=(10, 10), color="black"
        )
    else:
        if not finalized:
            cow.draw()
            screen.draw.text(
                "Garden happy for: " +
                str(time_elapsed) + " seconds",
                topleft=(10, 10), color="black"
            )
            if (not garden_happy):
                screen.draw.text(
                    "GARDEN UNHAPPY—GAME OVER!", color="black",
                    topleft=(10, 50)
                )
                finalized = True
```

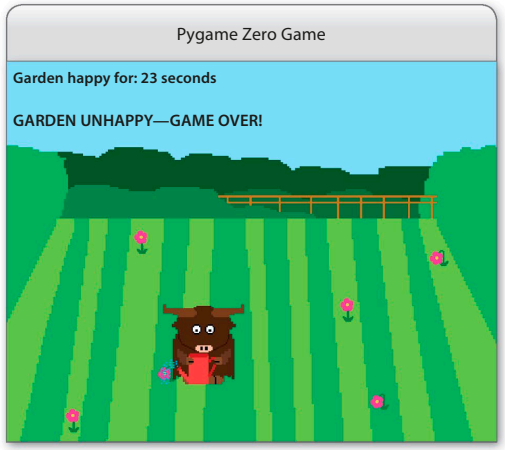
This displays a message that tells the player the game is over.



28 Test run

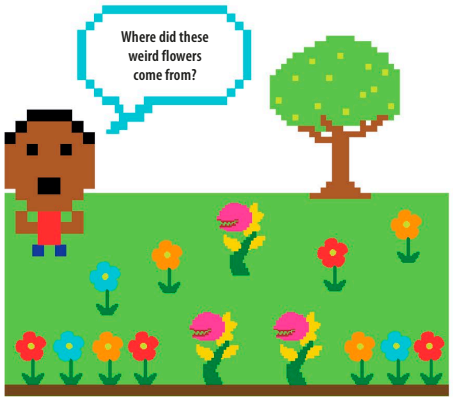
Save your IDLE file and run it from the command line in the Command Prompt or Terminal window. Try moving the cow around and water the wilted flowers. If a flower remains wilted for more than ten seconds, you will see a screen like the one shown here.





29 Added menace

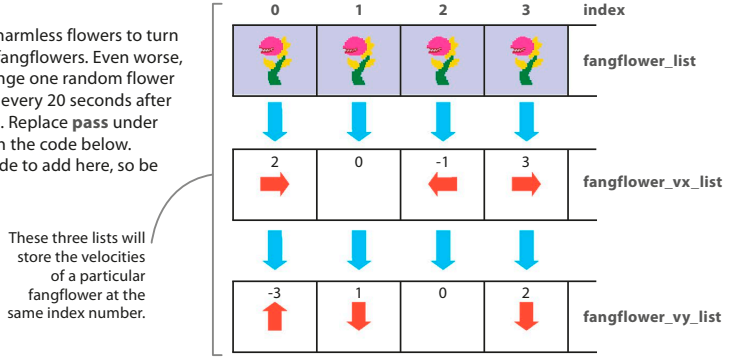
So far, keeping the garden happy has been difficult but not dangerous. What if the flowers start mutating into scary fangflowers that move around the garden trying to zap the cow? Let's add some functions to control the fangflower. Use placeholders for now and define them later on. Type this code above the **reset_cow()** function you defined in Step 21.




```
        index = index + 1
    return

def check_fangflower_collision():
    pass

def velocity():
    pass

def mutate():
    pass

def update_fangflowers():
    pass

def reset_cow():
    global game_over
    if not game_over:
        cow.image = "cow"
```


30 Mutation

It's time for your harmless flowers to turn into carnivorous fangflowers. Even worse, the code will change one random flower into a fangflower every 20 seconds after the first mutation. Replace **pass** under **def mutate()** with the code below. There's a lot of code to add here, so be extra careful.



These three lists will store the velocities of a particular fangflower at the same index number.

If the game is not over and there are still flowers left to mutate, this block of code will run.

These are the global variables needed in this function.

```
def mutate():
    global flower_list, fangflower_list, fangflower_vy_list
    global fangflower_vx_list, game_over
    if not game_over and flower_list:
        rand_flower = randint(0, len(flower_list) - 1)
        fangflower_pos_x = flower_list[rand_flower].x
        fangflower_pos_y = flower_list[rand_flower].y
        del flower_list[rand_flower]
        fangflower = Actor("fangflower")
        fangflower.pos = fangflower_pos_x, fangflower_pos_y
        fangflower_vx = velocity()
        fangflower_vy = velocity()
        fangflower = fangflower_list.append(fangflower)
        fangflower_vx_list.append(fangflower_vx)
        fangflower_vy_list.append(fangflower_vy)
        clock.schedule(mutate, 20)
    return
```

This line picks a random flower to mutate.

This line removes the mutated flower from the list of flowers.

This line sets the fangflower at the same position as the flower it mutated from.

This sets how fast the fangflower is moving left or right on the screen.

This sets how fast the fangflower is moving up or down on the screen.

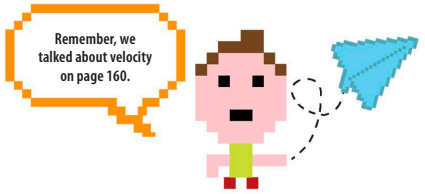
The fangflower's velocities are added to these lists.

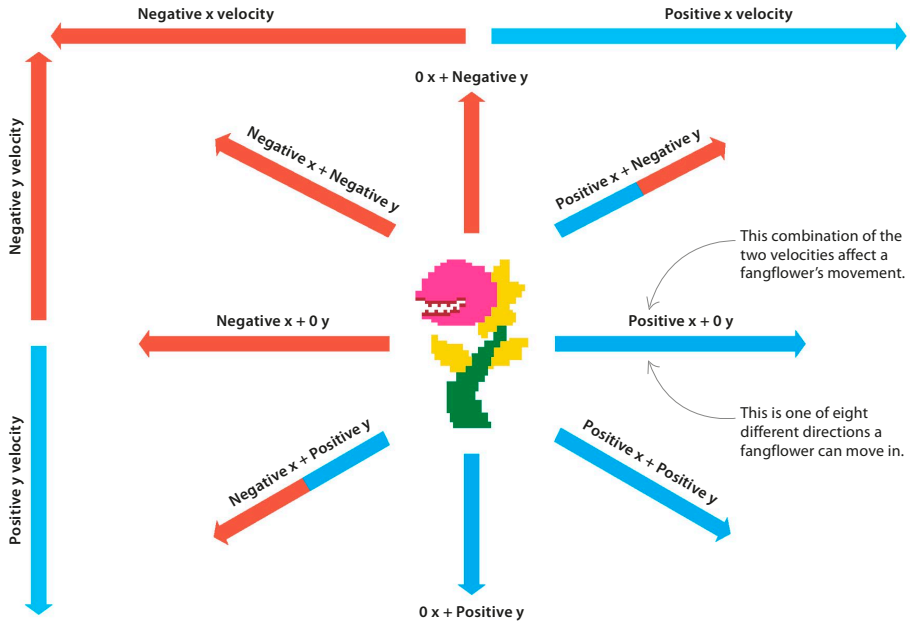
This line schedules a call to mutate a flower every 20 seconds.

This adds a new fangflower to the list of fangflowers.

31 Move the fangflower

Unlike other flowers, the fangflowers don't stay in one place. They move all over the garden trying to zap the cow. In this step, you'll add the code that generates the velocity of each fangflower along the x-axis and y-axis. The fangflowers will use a combination of these two velocities to move up, down, side to side, or diagonally. Add the following code under **def velocity()** from Step 29.





This combination of the two velocities affect a fangflower's movement.

This is one of eight different directions a fangflower can move in.

This generates the velocity of the fangflower with no direction yet.

If the direction is **0**, this returns a negative velocity.

```
def velocity():
    random_dir = randint(0, 1)
    random_velocity = randint(2, 3)
    if random_dir == 0:
        return -random_velocity
    else:
        return random_velocity
```

This line generates a number that represents the direction of the fangflower.

If the direction is **1**, this returns a positive velocity.

32 Update the fangflowers

It's time for the fangflowers to start moving. The code in this step will be called every time the **update()** function runs. It will also keep the fangflowers inside the garden by making them bounce off the edges of the game screen. Replace **pass** under **def update_fangflowers()** from Step 29 with this code. There's a lot of tricky code here, so type it in carefully.

These are the global variables used in this function.

```
def update_fangflowers():
    global fangflower_list, game_over
    if not game_over:
        index = 0
        for fangflower in fangflower_list:
            fangflower_vx = fangflower_vx_list[index]
            fangflower_vy = fangflower_vy_list[index]
            fangflower.x = fangflower.x + fangflower_vx
            fangflower.y = fangflower.y + fangflower_vy
            if fangflower.left < 0:
                fangflower_vx_list[index] = -fangflower_vx
            if fangflower.right > WIDTH:
                fangflower_vx_list[index] = -fangflower_vx
            if fangflower.top < 150:
                fangflower_vy_list[index] = -fangflower_vy
            if fangflower.bottom > HEIGHT:
                fangflower_vy_list[index] = -fangflower_vy
            index = index + 1
    return
```

This variable helps the program keep track of which item in the list it's dealing with.

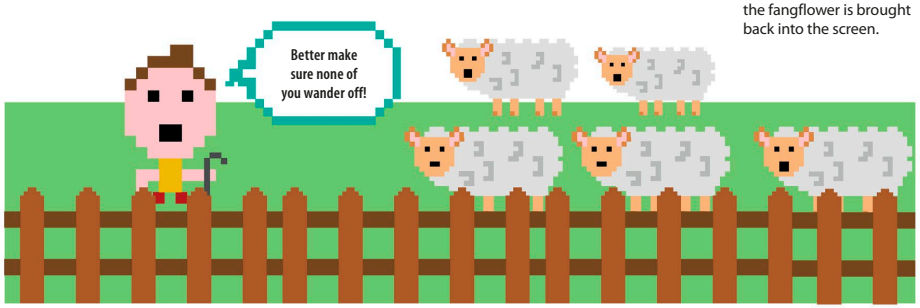
This loops over all the fangflowers in the list.

These get the x and y velocities of the fangflower.

These get the new position of the fangflower.

If the fangflower touches the left edge of the screen, this will make it start moving to the right.

By changing its y velocity, the fangflower is brought back into the screen.



33 Check for collisions

Now that your fangflowers are in motion, you need to add some code that will check if a fangflower has caught up with the cow to zap it! Replace **pass** in the **check_fangflower_collision()** function with the code shown below.

```
def check_fangflower_collision():
    global cow, fangflower_list, fangflower_collision
    global game_over
    for fangflower in fangflower_list:
        if fangflower.colliderect(cow):
            cow.image = "zap"
            game_over = True
            break
    return
```

These are the global variables used in this function.

This adds an image to show the cow has been zapped.

This tells the program that the game is over.

This checks if the fangflower and cow are next to each other.

This line stops the program from checking other fangflowers.

34 Drawing results

If a fangflower manages to zap the cow, it's game over. Add this code to the **draw()** function to display a game over message.

Don't forget to save your work.

```
        if (not garden_happy):
            screen.draw.text(
                "GARDEN UNHAPPY—GAME OVER!", color="black",
                topleft=(10, 100)
            )
            finalized = True
    else:
        screen.draw.text(
            "FANGFLOWER ATTACK—GAME OVER!", color="black",
            topleft=(10, 50)
        )
        finalized = True
    return
```

This block of code runs if the garden is still happy but the cow has been zapped.

This draws a message on the screen to show the game is over because of a fangflower attack.

This makes sure the code is not run again.

35 Set up the update

The fangflowers are ready to attack. The last thing you need to do is add some code that starts the whole mutation process if the garden has been happy for more than 15 seconds. Go to the **update()** function and add the code as shown here.

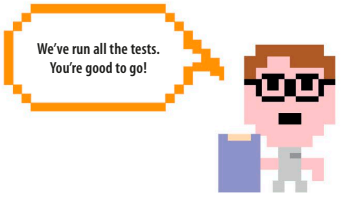
```
def update():
    global score, game_over, fangflower_collision
    global flower_list, fangflower_list, time_elapsed
    fangflower_collision = check_fangflower_collision()
    check_wilt_times()
    if not game_over:
        if keyboard.space:
            cow.image = "cow-water"
            clock.schedule(reset_cow, 0.5)
            check_flower_collision()
        if keyboard.left and cow.x > 0:
            cow.x -= 5
        elif keyboard.right and cow.x < WIDTH:
            cow.x += 5
        elif keyboard.up and cow.y > 150:
            cow.y -= 5
        elif keyboard.down and cow.y < HEIGHT:
            cow.y += 5
        if time_elapsed > 15 and not fangflower_list:
            mutate()
        update_fangflowers()
```

This checks if the garden has been happy for more than 15 seconds and if any fangflowers have appeared on the screen yet.

This line mutates a flower into a fangflower.

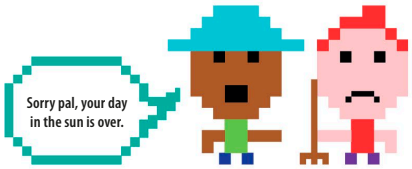
36 Test and play!

Green thumbs at the ready, your game can now be played! Save and run the file from the command line. Make sure the cow keeps the flowers watered while avoiding the dangerous fangflowers. What does your screen look like when the fangflowers appear and finally zap the cow?

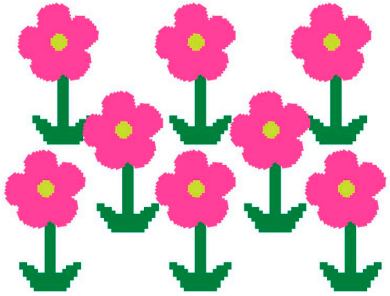


Hacks and tweaks

Do you want to make the game even more exciting? You can try out some of these ideas and add new features to the game.

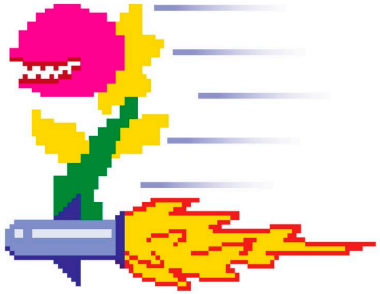


△ **Change the gardener**
You might think a cow is an unusual gardener. Why don't you look for another character in the Python Game Resource Pack? You could also make a new character using any 8-bit editor available online.



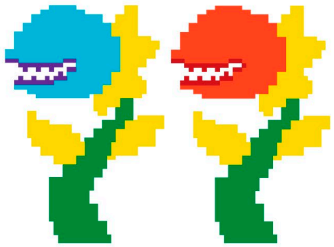
```
clock.schedule(add_flowers, 4)
```

△ **More flowers**
You can change how often a new flower appears on the screen to make the game easier or harder. Update the code under **def add_flowers()** to change how often it schedules a call to itself.



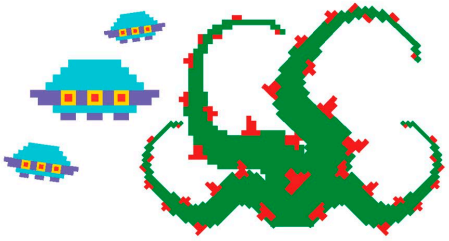
```
random_velocity = randint(2, 3)
```

△ **Faster fangflowers!**
You can make the fangflowers move faster by changing the possible range of **random_velocity**. Try increasing the range by using something like **randint(4, 6)**.

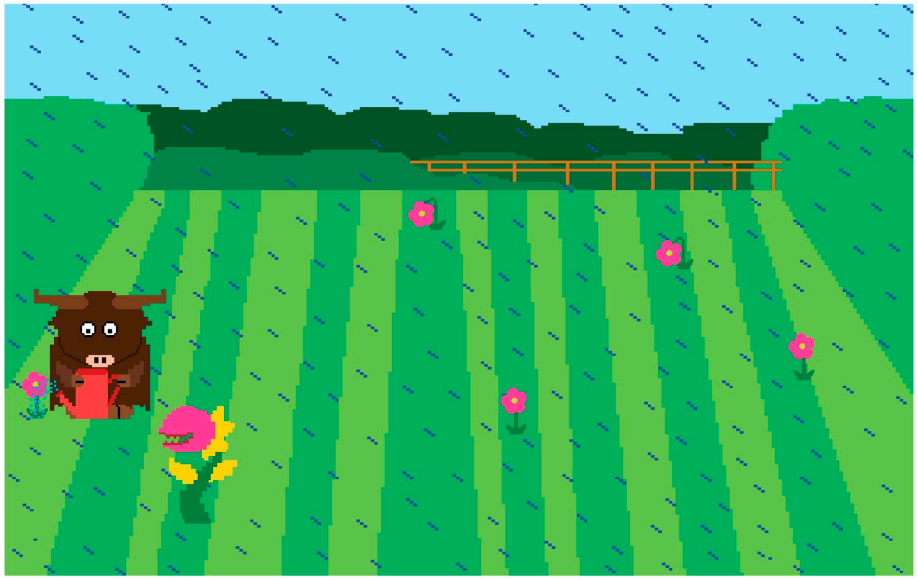


```
clock.schedule(mutate, 20)
```

△ **More fangflowers**
Is the game too hard or too easy? You can change the code in **mutate()** to make fangflowers appear more or less often.



◁ **Add new enemies**
If you find the game is not challenging enough at the moment, you can add more enemies. Use an online 8-bit editor to create more characters and then add some functions to control their behavior. These functions would be similar to the ones that control the fangflowers in the game. The new enemies could be flowers that fire pellets at the cow if it gets too close, thorn bushes that snake out long stems to catch the cow, or even aliens in flying saucers who are after the fangflowers.



△ **Rain in the garden**
What happens if it rains in the garden? The flowers will be much happier and wouldn't need watering. But they will also mutate into fangflowers more quickly. To make it look like it's raining, you can update the background with another image from the Python Games Resource Pack or create a new background on your own. To control the new background, create a new variable called **raining** and change the **draw()** function to update the background based on the variable's value.

```
if not raining:
    screen.blit("garden", (0, 0))
else:
    screen.blit("garden-raining", (0, 0))
```