Lesson 4 Simulating a Starry Sky

Throughout history, people have had a unique fascination with the stars. Gazing at the countless stars in the sky is like witnessing blinking eyes in the darkness. They resemble shining pearls, twinkling and sparkling, or like flickering silver lights on an endless ocean. It truly captivates the imagination.

In this lesson, we will embark on a hands-on experience to draw individual stars and make them float and drift randomly on the screen, creating a dynamic and vast starry sky simulation.

Task Objectives

Draw small stars on the screen and make them drift.

Knowledge points

1. Introduction to the turtle library
2. Learning how to draw stars using the turtle library
3. Learning how to make stars randomly appear and move using the turtle library

Material List

Hardware List:

UNIHIKER x1	Type-C & Micro Dual-Use USB Cable x1

Software Preparation: Mind+ Programming Softwarex1

Knowledge background

1. What is the turtle library?

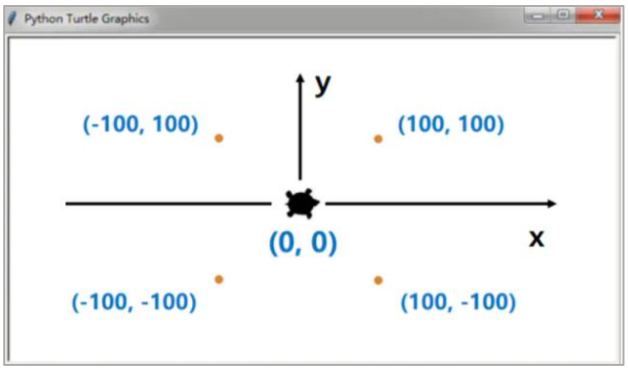
The turtle library, also known as the turtle graphics library, is a popular Python library used for drawing graphics. Its principle is to generate graphics by moving a pen along a path. We can use the turtle library to draw various shapes on the screen, such as rectangles, circles, and stars.

2. General knowledge of the turtle library

(1) Turtle coordinate system

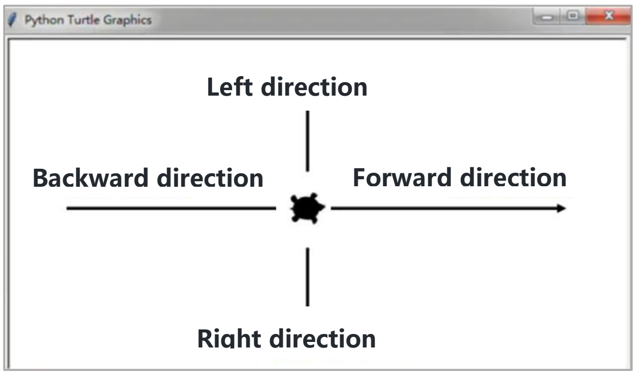
A. Absolute coordinates

The turtle coordinate system forms a four-quadrant coordinate system with the origin (0, 0) at the center of the screen.



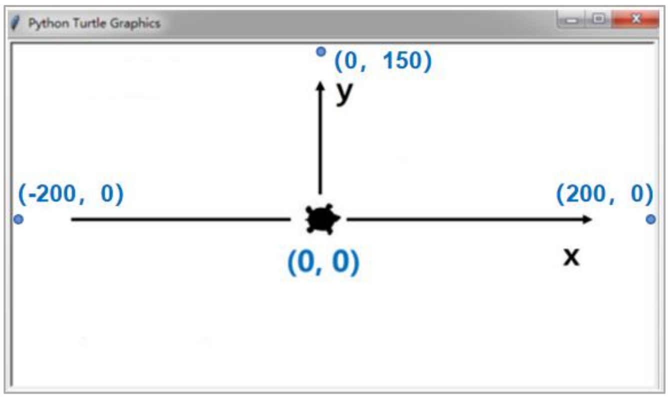
B. Turtle coordinates

Turtle coordinates are based on the turtle's point of view and are divided into four directions.



(2) Canvas

A canvas is like a blank piece of paper on which we can draw shapes of different colors, sizes, and thicknesses. Each shape drawn by the turtle is displayed on a canvas that is automatically generated. The default size of the canvas is (400, 300), where 400 represents the pixel value of the length and 300 represents the pixel value of the width. We can set the size and initial position of the canvas, and we can also treat the canvas as a coordinate system where the little turtle starts moving from the origin (0, 0) with each step.



(3) Three Key Elements of Turtle Graphics

There are three key elements in turtle graphics: direction, position, and pen. When creating turtle graphics, the main focus is to control these elements to draw the desired shapes.

Direction is mainly used to control the direction of the turtle's movement. Position is mainly used to control the distance the turtle moves. The pen mainly acts as the drawing tool. In turtle graphics, the pen can be used to control the thickness, color, and speed of the movement of the lines.

3. Common Functions for Screen Control in the Turtle Library

There are many functions related to screen control in the turtle library, but we will only use a portion of them. When programming, after importing the library with "import turtle", you can use the format "turtle.function_name()" to achieve the desired functionality. Tips: These functions are all methods of the TurtleScreen class.

(1) The setup() function is used to set the size and position of the drawing window (canvas)

With the setup() function, we can set the size and position of the current window screen. When the parameter is an integer, the unit is in pixels. If no parameter is entered, the default size will be used.

```
turtle.setup(240, 320)  # Create the window with the specified width and height
```

Here, 240 and 320 respectively refer to the width and height of the window, which is the same as the screen resolution mentioned in the previous sentence.

(2) The bgpic() function is used to set the background image of the screen

With the bgpic() function, we can set the background image for the current window screen.

```
turtle.bgpic('123.png')  # Add the background image of a galaxy
```

Here, "123.png" refers to the specific background image. In order to achieve a better display effect, we can adjust its size to be consistent with the screen.

(3) The tracer() function is used to set the animation on or off

With the tracer() function, we can set the drawing animation to be turned on or off. If we want to draw the graphics all at once without displaying the drawing process, we can choose to turn off the animation.

```
turtle.tracer(True)  # Turn on animation

turtle.tracer(False)  # Turn off animation
```

Here, "True" and "False" respectively refer to turning on and turning off the animation of drawing traces.

(4) The delay() function is used to set the drawing delay in the graphics

The delay() function can be used to set the drawing delay for the current window screen, measured in milliseconds. The longer the delay, the slower the animation, and setting it to 0 can avoid lagging.

```
turtle.delay(0)  # Set the drawing delay in milliseconds
```

Here, '0' refers to the specific time for the drawing delay on the canvas.

4. Common functions for pen movement in the turtle library

There are many common functions for pen movement in the turtle library. We only use a portion of them. When programming, we can use the format "turtle.function_name()" to achieve the desired functionality. Tip: All of these functions come from methods within the Turtle class.

(1) The function forward() controls the movement of the pen forward by a specified distance

By using forward() function, we can move the pen forward and draw lines accordingly.

```
turtle.forward(100) # Move the pen forward by 100 units
```

Here, "100" refers to a specific pixel value, which can be changed to adjust the length of the line. Similarly, we can use the "backward()" function to move the pen backwards and draw lines accordingly.

(2) The function left() controls the rotation of the pen to the left by a specified angle

The initial direction of the pen is to the right, and we can change the direction of the pen using the left() function.

```
turtle.left(90)  # Turn the pen left by 90 degrees
```

Here, "90" refers to the specific angle value, and we can change the direction by changing the angle value. Similarly, we can also use the "right()" function to rotate the pen to the right.

(3) The goto() function is used to move the turtle to a specific coordinate position

```
turtle.goto(110, 120) # Move to the coordinate position (110, 120)
```

"110" represents the horizontal coordinate of the position to which the turtle is to move, while "120" represents the corresponding vertical coordinate. By changing the horizontal and vertical coordinates, we can change the position to which the turtle moves. Similarly, we can also use the "set(x)" and "set(y)" functions to set the horizontal and vertical coordinates of the turtle pen separately.

(4) The speed() function sets the speed of the turtle's movements

The speed of the turtle's movements can be set using an integer value within the range of 0-10.

- "fastest": 0
- "fast": 10
- "normal": 6
- "slow": 3
- "slowest": 1

The speed setting from 1 to 10 determines the animation speed of the line drawing and turning movements, with higher values indicating faster animation. A speed of 0 means that no animation occurs. When moving forward or backward, the turtle will appear to jump to the new position, while turning left or right will result in an immediate change in direction.

```
self.speed(0)  # Set the speed of the turtle pen to the fastest
```

(5) The clone() function clones the pen object

After creating a pen object, we can use the clone() function to clone the object.

```
tt = turtle.Turtle(shape="turtle") # Instantiate the Turtle class, create an object with the shape of a turtle

star = tt.clone() # Clone the tt object to create a new object called star with the same shape, size, color, and other parameters
```

Here, we manually created a pen object with the shape of a turtle, and then used the clone() function to clone the object, which was stored in the variable "star".

Tips: The pen object in the turtle library is created by default and has the shape of a turtle. If we need to create additional pens and set their shape, we can do so by instantiating the Turtle class, as shown above.

5. Common functions for controlling the pen in the turtle library

There are many functions related to pen control in the turtle library, but we only use a portion of them. When programming, we can use the format "turtle.function_name()" to achieve the desired functionality.

(1) The shape() function displays the shape

The shape() function allows us to display the pen in various shapes, such as turtle, circle, arrow, etc., which are pre-installed in the turtle library's shape list. Of course, we can also manually draw our own shapes and save them to the shape list for display.

```
turtle.shape("turtle") # Set the shape of the turtle to "turtle"
```

Here, "turtle" refers to a specific turtle shape, and the actual shape can be changed by modifying the parameter inside the quotation marks.

Tips: We can use the "getshapes()" function to obtain all shapes in the turtle library's shape list.

(2) The register_shape() function is used to register a new shape in the turtle library

We can register shapes using the register_shape() function to add non-default shapes to the shape list.

```
turtle.register_shape("tt", shape)  # Add the drawn polygon to the shape list, named as "tt"
```

"tt" refers to the name of the shape to be added to the list, while "shape" refers to the specific shape to be added.

(3) The functions begin_poly(), end_poly(), and get_poly() are used to record a shape

In order to store the drawn shape as an item in the shape list, we need to record the vertices of the shape. In programming, we use the "begin_poly()" function to start recording the vertices before drawing the shape, and then use "end_poly()" to complete the recording after the shape is drawn. Finally, we use the "get_poly()" function to retrieve the recorded shape.

```
turtle.begin_poly()  # Begin recording the polygon's vertices

for i in range(5):  # Loop five times

    turtle.forward(15)  # Move forward by 15 units

    turtle.right(144)  # Turn right by 144 degrees

turtle.end_poly()  # End the polygon recording

shape = turtle.get_poly()  # Get the recorded polygon

turtle.register_shape("tt", shape)  # Register the polygon shape with the name "tt"
```

Here, we used the "begin_poly()", "end_poly()", and "get_poly()" functions to record the vertices of the drawn pentagon, and saved it as a basic shape in the shape list.

(4) The hideturtle() function hides the turtle pen

The function shape() allows the pen to display various shapes, while the function "hideturtle()" can hide it when we do not want to see it.

```
turtle.hideturtle() # Hide the turtle cursor
```

Similarly, we can make the pen reappear by using the "showturtle()" function.

(5) The "penup()" function lifts the pen

The "penup()" function lifts the pen, preventing it from drawing while moving.

```
turtle.penup() # Lift the pen up
```

Similarly, if we want the pen to draw while moving, we can use "pendown()" to put the pen back down.

(6) The "pencolor()" function sets the color of the pen

By default, the pen color is black when moving to draw. However, the "pencolor()" function can be used to set the color of the pen, thereby changing the color of the drawn shape.

```
turtle.pencolor('red') # Set the pen color to red
```

In this case, "red" refers to a specific color, which can also be represented by RGB values, hexadecimal values, or named color constants.

(7) The "fillcolor()", "begin_fill()", and "end_fill()" functions are used to fill the color of a shape.

The function pencolor() is used to set the color of the pen, while if we want to add a filling color to the drawn shape, we need to first set the filling color using the function "fillcolor()". Then, before starting to draw the shape, we use the function "begin_fill()" to initiate the filling, and finally, after the completion of the drawing, we use the function "end_fill()" to end the filling.

```
turtle.fillcolor('red') # Set the fill color to red.

turtle.begin_fill() # Start recording the vertices of the polygon.

for i in range(5): # Loop five times.

    turtle.forward(15) # Move forward 15 pixels.

    turtle.right(144) # Turn right 144 degrees.

turtle.end_fill() # End recording
```

Here, we achieved the red filling of the drawn pentagram using the three functions "fillcolor()", "begin_fill()", and "end_fill()".

(8) The functions "xcor()" and "ycor()" are used to detect and obtain the horizontal and vertical coordinates of the pen

To obtain the current coordinates of the pen, we can use the functions "xcor()" and "ycor()".

```
x = turtle.xcor() # Detects the horizontal coordinate of the pen's current position

y = turtle.ycor() # Detects the vertical coordinate of the pen's current position
```

Here, "x" and "y" are two variables used to store the current horizontal and vertical coordinates of the pen detected by the functions.

(9) The "done()" function is used to end the drawing process and keep the window open

After the completion of the drawing process, we can use the "done()" function to end the process and keep the drawing window open.

```
turtle.done() # Terminate the drawing and keep the window open
```

It is important to note that the "done()" function must be the last line of code in a turtle graphics program.

Hands-on practice

Task Description 1: Drawing a Star

Display a star drawn using the turtle library on the screen.

1. Hardware setup

STEP 1: Connect the UNIHIKER to the computer via a USB cable.

2. program coding

STEP 1: Create and Save Project File

Launch Mind+, save the project as "004 Simulating a Starry Sky".

STEP 2: Create and Save Python File

Create a Python program file named "main1.py" and double-click to open it.

STEP 3: Importing Images

To import the required PNG images into the project folder.

STEP 4: Programming

(1) Import necessary libraries

In this task, we need to use the turtle library to draw a star, so we need to import it first.

```
import turtle  # Import the turtle library
```

(2) Create a drawing window

To ensure that the screen size of the blank canvas remains consistent with the drawing canvas during the drawing process, we need to manually create a fixed-size canvas as the drawing window, import a background image into it, and set the drawing delay.

```
# Create the drawing window

width, height = 240, 320  # Set the width and height of the window

turtle.setup(width, height)  # Create the window with the specified width and height

turtle.bgpic('galaxy.png')  # Add the background image of a galaxy

turtle.delay(8)  # Set the drawing delay in milliseconds (the larger the value, the slower the an
imation)
```

(3) Create a pen and set the drawing parameters

Afterwards, we create a turtle pen, hide it so that it is not displayed during drawing, and set its movement speed, pen color, fill color, and other drawing-related parameters.

```
# Create a turtle pen and set its parameters

t = turtle.Turtle()  # Instantiate the Turtle class to create a turtle object as the pen

t.hideturtle()  # Hide the turtle pen

t.speed(0)  # Set the speed of the turtle pen to the fastest

t.pencolor("white")  # Set the color of the pen

t.fillcolor("white")  # Set the fill color
```

(4) Draw a star

Next, we can freely draw stars, the specific steps are as follows.

```
t.pendown()  # Lower the pen to start drawing

t.begin_fill()  # Begin the shape fill

# Drawing the star shape

for i in range(5):  # Loop five times

    t.forward(30)  # Move the turtle pen forward by 30 units

    t.right(144)  # Turn the turtle pen right by 144 degrees

t.end_fill()  # End the shape fill

t.penup()  # Lift the pen up



turtle.done()  # End the drawing and display the final result
```

Tips: The complete example program is as follows:

```
# Drawing a small star

import turtle  # Import the turtle library



# Create the drawing window

width, height = 240, 320  # Set the width and height of the window

turtle.setup(width, height)  # Create the window with the specified w
idth and height

turtle.bgpic('galaxy.png')  # Add the background image of a galaxy

turtle.delay(8)  # Set the drawing delay in milliseconds (the larger
the value, the slower the animation)



# Create a turtle pen and set its parameters

t = turtle.Turtle()  # Instantiate the Turtle class to create a turtl
e object as the pen

t.hideturtle()  # Hide the turtle pen

t.speed(0)  # Set the speed of the turtle pen to the fastest

t.pencolor("white")  # Set the color of the pen
```

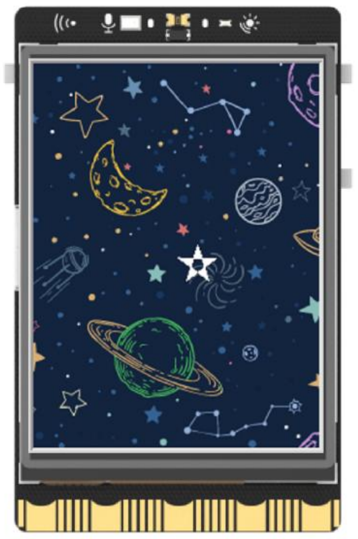
```
t.fillcolor("white")  # Set the fill color


t.pendown()  # Lower the pen to start drawing

t.begin_fill()  # Begin the shape fill

# Drawing the star shape

for i in range(5):  # Loop five times

    t.forward(30)  # Move the turtle pen forward by 30 units

    t.right(144)  # Turn the turtle pen right by 144 degrees

t.end_fill()  # End the shape fill

t.penup()  # Lift the pen up


turtle.done()  # End the drawing and display the final result
```

3. Running the Program

STEP 1: Remote connection to the UNIHIKER

STEP2: Running the Program and Observing the Results

Upon observing the UNIHIKER, one can first notice a vast and boundless background of the Milky Way, followed by a white pentagon star gradually appearing in the center of the screen.



Task Description 2: Drawing a Starry Sky

After successfully drawing a star, we can use this method to quickly draw multiple stars at any position on the screen.

1. program coding

STEP 1: Create and Save Project Files

Create a Python program file named "main2.py" and double-click to open it.

STEP 2: Programming

(1) Import necessary libraries

Here, since we will be randomly placing the stars, we need to import the random number library in addition.

```
import turtle  # Import the turtle library

import random  # Import the random library
```

(2) Creating the Drawing Window

```
# Create the drawing window

width, height = 240, 320  # Set the width and height of the window

turtle.setup(width, height)  # Create the window with the specified width and height

turtle.bgpic('galaxy.png')  # Add the background image of a galaxy

turtle.delay(8)  # Set the drawing delay in milliseconds (the larger the value, the slower the ani
mation)
```

(3) Creating the Star Class

Here, we create a Star class and instantiate it to randomly draw multiple stars, creating the effect of a starry sky. To begin, we set the basic properties of the pen, then draw a single star. We repeat this process 10 times to create the starry sky. Therefore, in the constructor method, we first set the turtle pen to be hidden and at the fastest moving speed. We then call the instance method for drawing the starry sky to achieve the desired effect.

```
# Define a Star class

class Star(turtle.Turtle):

    def __init__(self):

        turtle.Turtle.__init__(self)  # Unbind the methods from the Star class to also have the met
hods from the Turtle class
```

```python
        self.hideturtle()  # Hide the turtle pen
        self.speed(0)  # Set the speed of the turtle pen to the fastest
        self.draw_stars()  # Draw the entire starry sky
```

(4) Draw a single star

After that, we create an instance method to draw a single star, and the specific process is as follows.

```python
    def draw_star(self):  # Draw a single star
        self.pencolor("white")  # Set the color of the pen
        self.fillcolor("white")  # Set the fill color
        self.pendown()  # Lower the pen to start drawing
        self.begin_fill()  # Begin the shape fill
        # Drawing process
        for i in range(5):  # Loop five times
            self.forward(20)  # Move the turtle pen forward by 20 units
            self.right(144)  # Turn the turtle pen right by 144 degrees
        self.end_fill()  # End the shape fill
        self.penup()  # Lift the pen up
```

(5) Draw the starry sky

Next, we create an instance method to draw ten stars and randomly place them on the screen, and the specific process is as follows.

```python
    def draw_stars(self):  # Draw the entire starry sky
        self.penup()  # Lift the pen up
        # Drawing process
        for i in range(10):  # Loop ten times to draw 10 stars
            x = random.randint(-120, 120)  # Generate a random x coordinate within the screen range
            y = random.randint(-160, 160)  # Generate a random y coordinate within the screen range
            self.goto(x, y)  # Move to the initial position
            self.draw_star()  # Draw a single star
        self.end_fill()  # End the shape fill
        self.penup()  # Lift the pen up
```

(6) Instantiate the Star class and end the drawing

Finally, we create a complete starry sky by instantiating the Star class.

```
Star() # Create an instance of the Star class

turtle.done()  # End the drawing and keep the window open
```

Tips: The complete example program is as follows:

```
# Drawing a small star and making it appear randomly

import turtle  # Import the turtle library

import random  # Import the random library


# Create the drawing window

width, height = 240, 320  # Set the width and height of the window

turtle.setup(width, height)  # Create the window with the specified width and height

turtle.bgpic('galaxy.png')  # Add the background image of a galaxy

turtle.delay(8)  # Set the drawing delay in milliseconds (the larger the value, the slower the an
imation)


# Define a Star class

class Star(turtle.Turtle):

    def __init__(self):

        turtle.Turtle.__init__(self)  # Unbind the methods from the Star class to also have the met
hods from the Turtle class

        self.hideturtle()  # Hide the turtle pen

        self.speed(0)  # Set the speed of the turtle pen to the fastest

        self.draw_stars()  # Draw the entire starry sky


    def draw_star(self):  # Draw a single star

        self.pencolor("white")  # Set the color of the pen

        self.fillcolor("white")  # Set the fill color

        self.pendown()  # Lower the pen to start drawing
```

```python
        self.begin_fill()  # Begin the shape fill
        # Drawing process
        for i in range(5):  # Loop five times
            self.forward(20)  # Move the turtle pen forward by 20 units
            self.right(144)  # Turn the turtle pen right by 144 degrees
        self.end_fill()  # End the shape fill
        self.penup()  # Lift the pen up

    def draw_stars(self):  # Draw the entire starry sky
        self.penup()  # Lift the pen up
        # Drawing process
        for i in range(10):  # Loop ten times to draw 10 stars
            x = random.randint(-120, 120)  # Generate a random x coordinate within the screen range
            y = random.randint(-160, 160)  # Generate a random y coordinate within the screen range
            self.goto(x, y)  # Move to the initial position
            self.draw_star()  # Draw a single star
        self.end_fill()  # End the shape fill
        self.penup()  # Lift the pen up

# Create an instance of the Star class
Star()

turtle.done()  # End the drawing and keep the window open
```

2. Running the Program

STEP 1: Remote connection to the UNIHIKER

STEP2: Running the Program and Observing the Results

After clicking run, observe the blank space and you will see small stars randomly appearing on the screen.

Task Description 3: Simulating a Dynamic Starry Sky

Next, we will make the stars randomly appear at the top of the screen, and then slowly drift down, simulating a dynamic starry sky.

1. program coding

STEP1: Create and Save Python File

Create a Python program file named "main3.py" and double-click to open it.

STEP2: Programming

(1) Import necessary libraries

```
import turtle  # Import the turtle library

import random  # Import the random library
```

(2) Creating a Drawing Window

```
# Create the drawing window

width, height = 240, 320  # Set the width and height of the window

turtle.setup(width, height)  # Create the window with the specified width and height

turtle.bgpic('galaxy.png')  # Add the background image of a galaxy

turtle.delay(0)  # Set the drawing delay to 0 to avoid lagging
```

(3) Drawing the Stars and Recording their Outlines

In the previous task, we quickly drew 10 stars on the screen. However, in order to make the stars randomly drift, we need to treat them as independent objects. Therefore, we need to record the outlines of the drawn stars so that we can add them as shapes to the default shape list of the turtle library. This will allow us to present the stars as pen objects.

```
# Draw a star and record its outline
t = turtle.Turtle()  # Instantiate the Turtle class to create a turtle object as a pen
t.hideturtle()  # Hide the turtle pen
t.speed(0)  # Set the pen's speed to the fastest
t.pencolor("white")  # Set the pen color
t.fillcolor("white")  # Set the fill color
t.penup()  # Lift the pen up
t.goto(-140, 0)  # Move the pen outside the screen


turtle.tracer(False)  # Turn off animation
t.begin_fill()  # Start filling the shape
t.begin_poly()  # Start recording the vertices of the polygon
for i in range(5):  # Loop five times
    t.forward(15)  # Move the pen forward by 15 units
    t.right(144)  # Turn the pen right by 144 degrees
t.end_poly()  # End recording
t.end_fill()  # End filling
```

(4) Retrieve the recorded star outline and add it to the shape list

Next, we can freely draw stars by following these steps.

```
# Getting the recorded star outline and adding it to the shape list
shape = t.get_poly()  # Get the recorded polygon
turtle.register_shape("tt", shape)  # Add the drawn polygon to the shape list, named as "tt"
```

(5) Draw a star

Afterwards, we can call up the star shape as a pen object.

```
turtle.tracer(True)  # Turn on animation


# Create stars
tt = turtle.Turtle(shape='tt')  # Instantiate the Turtle class to create a pen object with the shap
```

```
e of a star and assign it to the variable "tt"
tt.pencolor("white")  # Set the pen color
tt.fillcolor("white")  # Set the fill color
tt.penup()  # Lift the pen up
tt.goto(-120, 160)  # Move the pen outside the screen
```

(6) Clone the star and make it randomly appear at the top of the screen

Next, we clone 50 star pen objects and make them randomly appear at the top of the screen.

```
stars = []  # Create a list to store stars
for s in range(50):  # Repeat 50 times
    star = tt.clone()  # Clone the star pen
    speed = random.random() / 3  # Generate a random speed
    star.speed(int(speed * 10))  # Set the movement speed of the star
    star.setx(random.randint(-width/2, width/2))  # Set the star's x-coordinate within the screen
    star.sety(height/2 + random.randint(1, height))  # Set the star's y-coordinate above the scre
en
    star.showturtle()  # Show the star
    stars.append(star)  # Add the star to the list
```

(7) Simulate the starry sky and make the stars fall randomly

Finally, we make them slowly fall down.

```
while True:  # Infinite loop
    for star in stars:  # Loop through the list of stars
        star.sety(star.ycor() - 3 * star.speed())  # Move the star vertically
        if star.ycor() < -300:  # Check if the star has moved below the screen
            star.hideturtle()  # Hide the star
            star.setx(random.randint(-width/2, width/2))  # Set the star's x-coordinate within the sc
reen
            star.sety(height/2 + random.randint(1, height))  # Set the star's y-coordinate above the
 screen
            star.showturtle()  # Show the star
```

Tips: The complete example program is as follows:

```python
# Making the star float from top to bottom

import turtle  # Import the turtle library

import random  # Import the random library


# Create the drawing window

width, height = 240, 320  # Set the width and height of the window

turtle.setup(width, height)  # Create the window with the specified width and height

turtle.bgpic('galaxy.png')  # Add the background image of a galaxy

turtle.delay(0)  # Set the drawing delay to 0 to avoid lagging


# Draw a star and record its outline

t = turtle.Turtle()  # Instantiate the Turtle class to create a turtle object as a pen

t.hideturtle()  # Hide the turtle pen

t.speed(0)  # Set the pen's speed to the fastest

t.pencolor("white")  # Set the pen color

t.fillcolor("white")  # Set the fill color

t.penup()  # Lift the pen up

t.goto(-140, 0)  # Move the pen outside the screen


turtle.tracer(False)  # Turn off animation

t.begin_fill()  # Start filling the shape

t.begin_poly()  # Start recording the vertices of the polygon

for i in range(5):  # Loop five times

    t.forward(15)  # Move the pen forward by 15 units

    t.right(144)  # Turn the pen right by 144 degrees

t.end_poly()  # End recording

t.end_fill()  # End filling


# Getting the recorded star outline and adding it to the shape list
```

```
shape = t.get_poly()  # Get the recorded polygon

turtle.register_shape("tt", shape)  # Add the drawn polygon to the shape list, named as "tt"

turtle.tracer(True)  # Turn on animation


# Create stars

tt = turtle.Turtle(shape='tt')  # Instantiate the Turtle class to create a pen object with the shap
e of a star and assign it to the variable "tt"

tt.pencolor("white")  # Set the pen color

tt.fillcolor("white")  # Set the fill color

tt.penup()  # Lift the pen up

tt.goto(-120, 160)  # Move the pen outside the screen


stars = []  # Create a list to store stars

for s in range(50):  # Repeat 50 times

    star = tt.clone()  # Clone the star pen

    speed = random.random() / 3  # Generate a random speed

    star.speed(int(speed * 10))  # Set the movement speed of the star

    star.setx(random.randint(-width/2, width/2))  # Set the star's x-coordinate within the screen

    star.sety(height/2 + random.randint(1, height))  # Set the star's y-coordinate above the scre
en

    star.showturtle()  # Show the star

    stars.append(star)  # Add the star to the list


while True:  # Infinite loop

    for star in stars:  # Loop through the list of stars

        star.sety(star.ycor() - 3 * star.speed())  # Move the star vertically

        if star.ycor() < -300:  # Check if the star has moved below the screen

            star.hideturtle()  # Hide the star

            star.setx(random.randint(-width/2, width/2))  # Set the star's x-coordinate within the s
creen

            star.sety(height/2 + random.randint(1, height))  # Set the star's y-coordinate above th
```

```
e screen

        star.showturtle()  # Show the star
```

2. Running the Program

STEP 1: Remote connection to the UNIHIKER

STEP2: Running the Program and Observing the Results

After clicking "run", observing the blank space, we can see that after waiting for a few seconds, small white stars fall randomly from the top of the screen, shining with the galaxy.

Challenge Yourself

Try changing the shape, size, quantity, and color of the stars, and see how the 3D starry sky effect changes!