Game Dev 1

Lesson 1 Dictionary

Activity 1:

```
#Create a dictionary
sample_dict = {
  "name": "Pulkit",
  "age": 23,
  "city": "Agra",
}

# Accessing values in a dictionary
print(sample_dict["name"])
print(sample_dict)

# Create a list with the same information to show the difference
between list and a dictionary
sample_list = ["Pulkit", 23, "Agra"]
print(sample_list[0])

# Get the list of keys
print(sample_dict.keys())

# Get the list of values
print(sample_dict.values())

for key in sample_dict.keys():
       print(key, sample_dict[key])

# Check if the key exists in the dictionary or not
if "country" in sample_dict:
       print(sample_dict["country"])
else:
       print("key does not exist")

# Add a key-value pair to the dictionary
sample_dict["Profession"] = "Software Engineer"
print(sample_dict)

# Delete a key-value pair
del(sample_dict["Profession"])
print(sample_dict)

# Change a value in the dictionary
sample_dict["city"] = "Bangalore"
print(sample_dict)

# Store a list as a value in the dictionary
sample_dict["marks"] = [99,87, 85, 92, 90]
print(sample_dict)

# Access a value in the list stored in the dictionary
print(sample_dict["marks][1])

# Create a nested dictionary
classroom = {
```

```
  "PulkitChawla" : {
    "age": 23,
    "marks": [89, 85, 90, 86, 90]
  },
  "Kanishk": {
    "age": 13,
    "marks": [90, 95, 85, 87, 80]
  }
}

# Go through basic dictionary operations for nested dictionary
print(classroom.keys())
print(classroom.values())

for i in classroom.keys():
       print(classroom[i])

classroom["PulkitChawla"]["age"] = 30
```

Activity 2:

```
#create an empty dictionary
countryDb={}
#infinite loop
while True:
    #print menu
    print("1. Insert")
    print("2. Display all countries")
    print("3. Display all capitals")
    print("4. Get capital")
    print("5. Delete")
    #get user choice
    choice=int(input("Enter your choice(1-5)"))
    #if insert
    if choice==1:
        country=input("Enter country :").upper()
        capital=input("Enter capital :").upper()
        countryDb[country]=capital
    #to display all countries
    elif choice==2:
        print(list(countryDb.keys()))
    #to display all capitals
    elif choice==3:
        print(list(countryDb.values()))
    #to display capital of a specific country
    elif choice==4:
        country=input("Enter country").upper()
        #print(countryDb[country])
        print(countryDb.get(country))
    #to delete entry of a specific country
    elif choice==5:
        country=input("Enter country :").upper()
        del countryDb[country]
    #if none of the above option
    else:
        break
```

Lesson 2 – Dictionary Continued

Activity 1:

#Panagram

```
# Count the occurrence of vowels in the string entered by the user

# Approach - 1
inputStr = input("Enter the string - ")
vowels = {
    "a":0,
    "e":0,
    "i":0,
    "o":0,
    "u":0
    }

for c in inputStr:
    if c in vowels:
        vowels[c] += 1

print(vowels)

# Approach - 2
inputStr = input("Enter the string - ")
vowelsList = []
vowels = {}

for c in inputStr:
    if c in vowelsList:
        if c in vowels:
            vowels[c] += 1
        else:
            vowels[c] = 1

print(vowels)

# Count the occurrence of each alphabhet in the string entered by the user

inputStr = input("Enter the string - ")
charCount = {}

for c in inputStr:
    if c isalpha():
        if c in charCount:
            charCount[c] += 1
        else:
            charCount = 1

print(charCount)



# Find if the number entered by the user is a Panagram or not ?

numberAsString = input("Enter the number - ")
```

```
numCount = {
    "1":0,
    "2":0,
    "3":0,
    "4":0,
    "5":0,
    "6":0,
    "7":0,
    "8":0,
    "9":0,
    "9":0
    }

for num in numberAsString:
    if num in numCount:
        numCount[num] += 1

panagram = True
for count in numCount.values():
    if count == 0:
        panagram = False

if panagram:
    print("Entered number is a Panagram")
else:
    print("Entered number is not a Panagram")
```

Lesson 3 – PgZero and Shapes – install VSC and Python

```
import pgzrun
from random import randint

WIDTH = 300
HEIGHT = 300
def draw():
    r = 255
    g = 0
    b = randint(120, 255)

    width = WIDTH
    height = HEIGHT - 200

    for i in range(20):
        rect = Rect((0, 0), (width, height))
        rect.center = 150, 150
        screen.draw.rect(rect, (r, g, b))

        r -= 10
        g += 10

        width -= 10
        height += 10


pgzrun.go()
```

Lesson 4 – Shoot the alien - Mouse click event

```
# Import the Pygame Zero Library
import pgzrun
from random import randint

# Pygame Standard for deciding the title of your game window
TITLE = "Good Shot"
# Pygame Standard for deciding the width and height for your game
window in pixels
WIDTH = 500
HEIGHT = 500

# variable to store the message displayed on your screen
message = ""

# Actor is built-in object in pgzero
# interacts with other actors, move around on screen
# Similar to Sprite in Scratch
alien = Actor('alien')
#alien.pos = 50,50


# Default function which will be called to update the screen
def draw():
  # screen is another built-in object
  screen.clear()
  screen.fill(color = (128, 0, 0))
  # place_alien()
  alien.draw()
  screen.draw.text(message, center = (400, 10), fontsize= 30)

def place_alien():
  alien.x = randint(50, WIDTH-50)
  alien.y = randint(50, WIDTH-50)

def on_mouse_down(pos):
  #print("Hello World")
  global message
  if alien.collidepoint(pos):
    message = "Good Shot"
    place_alien()
  else:
    message = "You missed"

# This method needs to be called to start processing
place_alien()
pgzrun.go()
```

Lesson 5 – Lists

```
# Create a 2D - List
matrix = [[1,2,3], [4,5,6], [7,8,9]]

print(matrix)

# Number of rows
print(len(matrix)

# Number of columns
print(len(matrix[0]))

# Accessing a element in 2D List
print(matrix[1][2])

# Looping through values in the 2D List
for i in range(0, len(matrix)):
    for j in range(0, len(matrix[0])):
        print(matrix[i][j], end = " ")
    print("\n")

# Take an input for a matrix and print the elements

rows = int(input("Enter the number of rows - "))
columns = int(input("Enter the number of columns - "))

matrix = []

for i in range(rows):
    temp = []
    for j in range(columns):
        x = int(input("Enter your element - "))
        temp.append(x)
    matrix.append(temp)

for i in range(rows):
    for j in range(columns):
        print(matrix[i][j], end = " ")
    print("\n")


# Take the square-matrix as input and print the diagonal elements

n = int(input("Enter the dimensions of the matrix - "))

for i in range(n):
    temp = []
    for j in range(n):
        x = int(input("Enter your element - "))
        temp.append(x)
    matrix.append(temp)

for i in range(n):
    print(matrix[i][i])
```

 # Program for addition and subtraction and subtraction of 2 2D Lists

```
matrixA = [[1,2], [3,4]]
matrixB = [[5,6], [7,8]]

additionResult = [[0,0], [0,0]]
subtractionResult = [[0,0], [0,0]]

for i in range(0,2):
    for j in range(0,2):
        additionResult[i][j] = matrixA[i][j] + matrixB[i][j])
        subtractionResult[i][j] = matrixA[i][j] - matrixB[i][j])

# Addition Result
for i in range(2):
    for j in range(2):
        print(additionResult[i][j], end = " ")
    print("\n")

# Subtraction Result
for i in range(2):
    for j in range(2):
        print(subtractionResult[i][j], end = " ")
    print("\n")
```

 # Optional - Program for multiplication of matrices

```
matrixA = [[1,2], [3,4]]
matrixB = [[5,6], [7,8]]

result = [[0,0], [0,0]]

for i in range(0,2):
  for j in range(0,2):
    for k in range(0,2):
      result[i][j] = result[i][j] + (matrixA[i][k] * matrixB[k][j])

for i in range(2):
    for j in range(2):
        print(result[i][j], end = " ")
    print("\n")
```

Lesson 6 – Bumblebee and the flower - Keyboard events

```
import pgzrun
from random import randint
WIDTH = 600
HEIGHT = 500
score = 0
game_over = False

bee = Actor("bee")
bee.pos = 100,100
flower = Actor("flower")
flower.pos = 200,200

def draw():
    screen.blit("background", (0,0))
    flower.draw()
    bee.draw()
    screen.draw.text("Score: " + str(score), color="black",
topleft=(10,10))

    if game_over:
            screen.fill("pink")
            screen.draw.text("Time's Up! Your Final Score: " +
str(score), midtop=(WIDTH/2,10),
            fontsize=40, color="red")


def place_flower():
    flower.x = randint(70, (WIDTH-70))
    flower.y = randint(70, (HEIGHT-70))

def time_up():
    global game_over
    game_over = True

def update():
    global score

    if keyboard.left:
        bee.x = bee.x - 2
    if keyboard.right:
        bee.x = bee.x + 2
    if keyboard.up:
        bee.y = bee.y - 2
    if keyboard.down:
        bee.y = bee.y + 2

    flower_collected = bee.colliderect(flower)

    if flower_collected:
        score = score + 10
        place_flower()

clock.schedule(time_up, 60.0)

pgzrun.go()
```

Lesson 7 – Tuples

```
stuDetails=('Surabhi', 89)

#Packing
address = ('227', 'Brickfield Shelters', 'Bangalore', 'Karnataka',
'562107')

for x in address:
    print (x, end = ' ')


#Unpacking
houseno, apartName, city, state, pin = address

print()
print('HNO', houseno)
print('APT NO ', apartName)
print(city)
print(state)
print(pin)

#A tuple can also be created without using parentheses
my_tuple = 3, 4.6, "dog"
print(my_tuple)

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
print(n_tuple[0][3])       # 's'
print(n_tuple[1][1])       # 4

#Activity for the kids
# Accessing tuple elements using slicing
my_tuple = ('p','r','o','g','r','a','m','i','z')

# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:-7])

# elements 8th to end
# Output: ('i', 'z')
print(my_tuple[7:])

# elements beginning to end
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple[:])

# Changing tuple values
my_tuple = (4, 2, 3, [6, 5])
```

```
# TypeError: 'tuple' object does not support item assignment
# my_tuple[1] = 9

# However, item of mutable element can be changed
my_tuple[3][0] = 9    # Output: (4, 2, 3, [9, 5])
print(my_tuple)

# Tuples can be reassigned
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple)
```

Lesson 8 – Connecting satellites - Sprite interactions

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

WIDTH = 800
HEIGHT = 600

satellites = []
lines = []
next_satellite = 0

start_time = 0
total_time = 0
end_time = 0

number_of_satellite = 8

def create_satellites():
    global start_time
    for count in range(0, number_of_satellite):
        satellite = Actor("satellite")
        satellite.pos = randint(40, WIDTH-40), randint(40, HEIGHT-40)
        satellites.append(satellite)
    start_time = time()


def draw():
    global total_time

    screen.blit("background", (0,0))
    number = 1
    for satellite in satellites:
        screen.draw.text(str(number), (satellite.pos[0],
satellite.pos[1]+20))
        satellite.draw()
        number = number + 1

    for line in lines:
        screen.draw.line(line[0], line[1], (255,255,255))

    if next_satellite < number_of_satellite:
        total_time = time() - start_time
```

```
        screen.draw.text(str(round(total_time,1)), (10,10),
fontsize=30)
    else:
        screen.draw.text(str(round(total_time,1)), (10,10),
fontsize=30)


def update():
    pass

def on_mouse_down(pos):
    global next_satellite, lines

    if next_satellite < number_of_satellite:
        if satellites[next_satellite].collidepoint(pos):
            if next_satellite:
                lines.append((satellites[next_satellite-1].pos,
satellites[next_satellite].pos))
            next_satellite = next_satellite + 1
        else:
            lines = []
            next_satellite = 0

create_satellites()

pgzrun.go()
```

Lesson 9 – Sets

```
# Converting a list to a set

sample_list = [1,1,2,2,3,3]

sample_set = set(sample_list)

print(sample_set)

# Show that sets are not indexable
print(sample_set[2])

# Check if an element exists in the set
if 4 in sample_set:
    print("Yes")
else:
    print("No")

# Adding element to the set
myset = set([])
myset.add(3)
myset.add(3)
myset.add(2)
myset.add(1)

print(myset)
```

```
# Remove the elements from the set,

myset.remove(2)
# Throws error if element is not present
myset.remove(5)
# Does not throw error if element is not present
myset.discard(5)

print(myset)

#Set Operations
# 1) Union
# 2) Intersection
# 3) Difference
# 4) Symmetric Difference

"""
a = {1,2,3,4,5}
b = {4,5,6,7,8}

Union means addition of sets
a U b = {1,2,3,4,5,6,7,8}

Internsection means the common elements between two sets
a intersection B = {4,5}
c = {1,2,3}
d = {4,5,6}
c intersection d = None

difference of A and B is the elements that exist in A but not in B
a = {1,2,3,4,5}
b = {4,5,6,7,8}
a - b = {1,2,3}
b - a = {6,7,8}

Symmetric difference is (union of sets - intersection of sets)
a = {1,2,3,4,5}
b = {4,5,6,7,8}
a symDiff b = {1,2,3,6,7,8}

"""

a = {1,2,3,4,5}
b = {4,5,6,7,8}

# Union of Sets
print(a.union(b))
print(a | b)

# Intersection of Sets
print(a.intersection(b))
print(a & b)

# Difference of Sets
print(a.difference(b))
print(a - b)
```

```
# Symmetric Difference of Sets
print(a.symmetric_difference(b))
print(a ^ b)
```

Lesson 10 – Shooting stars

```
import pgzrun
import random

FONT_COLOR = (255,255,255)
WIDTH = 800
HEIGHT = 600
CENTRE_X = WIDTH / 2
CENTRE_Y = HEIGHT / 2
CENTRE = (CENTRE_X, CENTRE_Y)
FINAL_LEVEL = 6
START_SPEED = 10
COLORS = ["blue","green","orange","purple","yellow"]


game_over = False
game_complete = False
current_level = 1
stars = []
animations = []

def draw():
    global stars, current_level, game_over, game_complete
    screen.clear()
    screen.blit("space", (0,0))
    if game_over:
        display_message("GAME OVER","Try again.")
    elif game_complete:
        display_message("YOU WON!","Well done.")
    else:
        for star in stars:
            star.draw()
    # if game_over or game_complete:
    #     clock.unschedule(shuffle_stars)


# count = 0
def update():
    global stars, count
    if len(stars) == 0:
        stars = make_stars(current_level)
    # else:
    #     count = count + 1
    #     if count % 30 == 0:
    #         layout_stars(stars)
    #         count = 0


def make_stars(number_of_extra_stars):
    colors_to_create = get_color_to_create(number_of_extra_stars)
    new_stars = create_stars(colors_to_create)
    layout_stars(new_stars)
```

```
    animate_stars(new_stars)
    return new_stars


def get_color_to_create(number_of_extra_stars):
    colors_to_create = ["red"]
    for i in range(0, number_of_extra_stars):
        random_color = random.choice(COLORS)
        colors_to_create.append(random_color)
    return colors_to_create

def create_stars(colors_to_create):
    new_stars = []
    for color in colors_to_create:
        star = Actor(color + "-star")
        new_stars.append(star)
    return new_stars

def layout_stars(stars_to_layout):
    number_of_gaps = len(stars_to_layout) + 1
    gap_size = WIDTH / number_of_gaps
    random.shuffle(stars_to_layout)
    for index, star in enumerate(stars_to_layout):
        new_x_pos = (index + 1) * gap_size
        star.x = new_x_pos


def animate_stars(stars_to_animate):
    global animations
    for star in stars_to_animate:
        duration = START_SPEED - current_level
        star.anchor = ("center","bottom")
        animation = animate(star, duration=duration,
on_finished=handle_game_over, y=HEIGHT)
        animations.append(animation)

def handle_game_over():
    global game_over
    game_over = True

def on_mouse_down(pos):
    global stars, current_level
    for star in stars:
        if star.collidepoint(pos):
            if "red" in star.image:
                red_star_click()
            else:
                handle_game_over()

def red_star_click():
    global current_level, stars, animations, game_complete
    stop_animations(animations)
    if current_level == FINAL_LEVEL:
        game_complete = True
    else:
        current_level = current_level + 1
        stars = []
        animations = []
```

```
def stop_animations(animations_to_stop):
    for animation in animations_to_stop:
        if animation.running:
            animation.stop()


def display_message(heading_text, sub_heading_text):
    screen.draw.text(heading_text, fontsize=60, center=CENTRE,
color=FONT_COLOR)
    screen.draw.text(sub_heading_text,fontsize=30,
center=(CENTRE_X,CENTRE_Y+30), color=FONT_COLOR)


# def shuffle_stars():
#     if stars:
#         layout_stars(stars)

# clock.schedule_interval(shuffle_stars, 0.5)

pgzrun.go()
```

Lesson 11 – Quiz Master

```
import pgzrun

TITLE = "Quiz Master"
WIDTH = 870
HEIGHT = 650

marquee_box = Rect(0,0,880,80)
question_box = Rect(0,0,650,150)
timer_box = Rect(0,0,150,150)
answer_box1 = Rect(0,0,300,150)
answer_box2 = Rect(0,0,300,150)
answer_box3 = Rect(0,0,300,150)
answer_box4 = Rect(0,0,300,150)
skip_box = Rect(0,0,150,330)

score = 0
time_left = 10
question_file_name = "questions.txt"
marquee_message = ""
is_game_over = False

answer_boxes = [answer_box1,answer_box2,answer_box3,answer_box4]
questions = []
question_count = 0
question_index = 0

marquee_box.move_ip(0,0)
question_box.move_ip(20,100)
timer_box.move_ip(700,100)
answer_box1.move_ip(20,270)
answer_box2.move_ip(370,270)
answer_box3.move_ip(20,450)
answer_box4.move_ip(370,450)
```

```
skip_box.move_ip(700,270)

def draw():
    global marquee_message
    screen.clear()
    screen.fill(color="black")
    screen.draw.filled_rect(marquee_box, "black")
    screen.draw.filled_rect(question_box, "navy blue")
    screen.draw.filled_rect(timer_box, "navy blue")
    screen.draw.filled_rect(skip_box, "dark green")

    for answer_box in answer_boxes:
        screen.draw.filled_rect(answer_box, "dark orange")

    marquee_message = "Welcome To Quiz Master..."
    marquee_message = marquee_message + f"Q: {question_index} of
{question_count}"

    screen.draw.textbox(marquee_message, marquee_box, color="white")
    screen.draw.textbox(
        str(time_left),timer_box,
        color="white", shadow=(0.5, 0.5),
        scolor="dim grey"
    )
    screen.draw.textbox(
        "Skip", skip_box,
        color="black", angle=-90
    )
    screen.draw.textbox(
        question[0].strip(), question_box,
        color="white", shadow=(0.5,0.5),
        scolor="dim grey"
    )
    index = 1
    for answer_box in answer_boxes:
        screen.draw.textbox(question[index].strip(), answer_box,
color="black")
        index = index + 1


def update():
    move_marquee()


def move_marquee():
    marquee_box.x = marquee_box.x - 2
    if marquee_box.right < 0:
        marquee_box.left = WIDTH


def read_question_file():
    global question_count, questions
    q_file=open(question_file_name, "r")
    for question in q_file:
        questions.append(question)
        question_count = question_count + 1
    q_file.close()
```

```
def read_next_question():
    global question_index
    question_index = question_index + 1
    return questions.pop(0).split(",")



def on_mouse_down(pos):
    index = 1
    for box in answer_boxes:
        if box.collidepoint(pos):
            if index is int(question[5]):
                correct_answer()
            else:
                game_over()
        index = index + 1

    if skip_box.collidepoint(pos):
        skip_question()


def correct_answer():
    global score, question, time_left, questions
    score = score + 1
    if questions:
        question = read_next_question()
        time_left = 10
    else:
        game_over()


def game_over():
    global question, time_left, is_game_over
    message = f"Game over!\nYou got {score} questions correct!"
    question = [message, "-","-","-","-",5]
    time_left = 0
    is_game_over = True


def skip_question():
    global question, time_left
    if questions and not is_game_over:
        question = read_next_question()
        time_left = 10
    else:
        game_over()


def update_time_left():
    global time_left
    if time_left:
        time_left = time_left - 1
    else:
        game_over()


read_question_file()
```

```
question = read_next_question()
clock.schedule_interval(update_time_left, 1)

pgzrun.go()
```

Lesson 12 – Gallaga Game

```
import pgzrun
import random
#screen dimensions
WIDTH = 1200
HEIGHT = 600

#definiting colours
WHITE = (255,255,255)
BLUE = (0,0,255)

#create a ship
ship = Actor('galaga')
bug = Actor('bug')

ship.pos = (WIDTH//2, HEIGHT-60)

speed = 5

#define a list for bullets
bullets = []

#defining a list of enemies
enemies = []

#we want 8 enemies
for x in range(8):
    for y in range(4):
        enemies.append(Actor('bug'))
        #now the enemies will be ina straight line
        enemies[-1].x = 100+ 50*x
        #starting off the screen thats why putting it at -100,
        #slowly the enemy will come down
        enemies[-1].y = 80 + 50*y

score = 0
direction = 1
ship.dead = False
ship.countdown = 90

#for updating the score
def displayScore():
    screen.draw.text(str(score), (50,30))

def gameOver():
    screen.draw.text("GAME OVER", (250,300))

def on_key_down(key):
    if ship.dead == False:
        if key == keys.SPACE:
            bullets.append(Actor('bullet'))
```

```
            #the last bullet added , set its position
            bullets[-1].x = ship.x
            bullets[-1].y = ship.y - 50


def update():
    global score
    global direction
    moveDown = False

    #move the ship left or right with arrow keys
    if ship.dead == False:
        if keyboard.left:
            ship.x -= speed
            if ship.x <= 0:
                ship.x = 0

        elif keyboard.right:
            ship.x += speed
            if ship.x >= WIDTH:
                ship.x = WIDTH
    #to fire bullets
    #it should not be while you on hold spapce key event
    #rather it should be on s[ace key down event
    '''
    if keyboard.space:
        print("Pressing space")
        bullets.append(Actor('bullet'))
        #the last bullet added , set its position
        bullets[-1].x = ship.x
        bullets[-1].y = ship.y
    '''
    for bullet in bullets:
        #if the bullet reaches the top of the screen it should get
removed
        #else the list will become huge
        if bullet.y <=0 :
            bullets.remove(bullet)
        else:
            bullet.y -= 10

    #check the position of the last enemy

    if len(enemies) == 0:
        gameOver()

    if len(enemies)>0  and (enemies[-1].x > WIDTH-80 or enemies[0].x <
80):
        moveDown = True
        direction = direction*-1
    for enemy in enemies:
        enemy.x += 5*direction
        if moveDown == True:
            enemy.y += 100
        if enemy.y > HEIGHT :
            enemies.remove(enemy)

        #checking if the enemy hits a bullet while moving down
```

```
        #iterate over all the bullets and check for a collision
        for bullet in bullets :
            if enemy.colliderect(bullet):
                score +=100
                #we also want to destory the bullet
                bullets.remove(bullet)
                #instead of removing the enemy we could send it back
up?
                enemies.remove(enemy)
                if len(enemies) == 0:
                    gameOver()
        #checking for enemy hits the ship
        if enemy.colliderect(ship):
            ship.dead = True
    if ship.dead:
        ship.countdown -=1
    if ship.countdown == 0:
        ship.dead = False
        ship.countdown = 90


def draw():
    screen.clear()
    screen.fill(BLUE)
    #ship.draw()
    for bullet in bullets:
        bullet.draw()
    for enemy in enemies:
        enemy.draw()
    #ship to be drawn last
    if ship.dead == False:
        ship.draw()
    displayScore()
    if len(enemies) == 0:
        gameOver()


pgzrun.go()
```

Lesson 13 – Ship

```
"""An example of using animate() and clock scheduling to move actors
around.

There are two actors in this example, each with a different movement
strategy.

The block
---------

The block moves in a loop around the screen:

* We schedule the move_block() function to be called every 2 seconds
using
  clock.schedule_interval().
```

```
* The next position of the block is given by calling next() on a
"cycle"
  object, returned by itertools.cycle(). This will cycle through the
block
  coordinates we provide it, repeating without end.
* We use animate() to move the block.


The ship
--------

The ship moves in a random dance in the middle of the screen. The ship
flips back and forth between a rotation phase and a movement phase:


* next_ship_target(): pick a new target location for the ship at
random, and
  animate rotating the ship to aim at it. When the rotation animation
is
  complete, we will call move_ship().
* move_ship(): Move the ship to its target. When the move animation is
  complete, we will call next_ship_target().


"""
import random
import itertools
import pgzrun


WIDTH = 400
HEIGHT = 400


# Define four sets of coordinates for the block to move between
BLOCK_POSITIONS = [
    (350, 50),
    (350, 350),
    (50, 350),
    (50, 50),
]
# The "cycle()" function will let us cycle through the positions
indefinitely
block_positions = itertools.cycle(BLOCK_POSITIONS)


block = Actor('block', center=(50, 50))
ship = Actor('ship', center=(200, 200))


def draw():
    screen.clear()
    block.draw()
    ship.draw()


# Block movement
# --------------
```

```
def move_block():
    """Move the block to the next position over 1 second."""
    animate(
        block,
        'bounce_end',
        duration=1,
        pos=next(block_positions)
    )


move_block()  # start one move now
clock.schedule_interval(move_block, 2)  # schedule subsequent moves


# Ship movement
# -------------

def next_ship_target():
    """Pick a new target for the ship and rotate to face it."""
    x = random.randint(100, 300)
    y = random.randint(100, 300)
    ship.target = x, y

    target_angle = ship.angle_to(ship.target)

    # Angles are tricky because 0 and 359 degrees are right next to
each other.
    #
    # If we call animate(angle=target_angle) now, it wouldn't know
about this,
    # and will simple adjust the value of angle from 359 down to 0,
which means
    # that the ship spins nearly all the way round.
    #
    # We can always add multiples of 360 to target_angle to get the
same angle.
    # 0 degrees = 360 degrees = 720 degrees = -360 degrees and so on.
If the
    # ship is currently at 359 degrees, then having it animate to 360
degrees
    # is the animation we want.
    #
    # Here we calculate how many multiples we need to add so that any
rotations
    # will be less than 180 degrees.
    target_angle += 360 * ((ship.angle - target_angle + 180) // 360)

    animate(
        ship,
        angle=target_angle,
        duration=0.3,
        on_finished=move_ship,
    )


def move_ship():
    """Move the ship to the target."""
```

```
    anim = animate(
        ship,
        tween='accel_decel',
        pos=ship.target,
        duration=ship.distance_to(ship.target) / 200,
        on_finished=next_ship_target,
    )


next_ship_target()
pgzrun.go()
```