Intro to Pygame

Mohamed Aturban

Contents

1 Midterm Exam 2: Due November 28, 2023, at 11:59 PM

For Midterm Exam 2, you are required to submit answers to the questions outlined in the following sections. The total points for this exam, including extra credit, amount to 130 points (100 standard points + 30 extra points). Please name your script files according to the descriptions in each question. When submitting scripts, include all necessary files, such as images and sound files, if applicable. Please ignore the small images above each question as they do not have any specific meaning; they are merely an easy way to remember what each question is about.

- Section 6.9 (8 + 5 + 8 points)
- Section 7.5 (8 × 4 points)
- Section 8.4 (2 × 10 Extra Points) (Optional)
- Section 9.5 (9 × 3 points)
- Section 14.1 (10 Extra Points) (Optional)
- Section 14.2 (10 points)
- Section 15.3 (10 points)

2 Installing Pygame

Pygame is a set of Python modules designed for writing video games. To install Pygame on a Windows machine, follow these steps:

1. Ensure you have Python installed on your system. If not, download and install Python from `https://www.python.org/downloads/`.
2. Open your command prompt (CMD).
3. Type the following command and press Enter:
   `pip install pygame`
4. Wait for the installation to complete.

To verify that Pygame has been installed correctly, you can run a simple test:

1. Open your Python IDE or the Python interactive shell.
2. Type the following commands:

   ```
   import pygame
   pygame.init()
   # to print the Pygame version
   print(pygame.ver)
   ```

3. If Pygame has been installed correctly, you should see the version number of Pygame printed without any errors, which means you can start developing games using Python.

3 Pygame Essentials: From Initialization to Game Loop

There are essential steps included in every Pygame program, which will be thoroughly explained in the following sections, specifically from Section 3.1 to Section 3.3.

3.1 Initialization

Every Pygame program begins by initializing the necessary modules for game development. This step ensures all the functionalities of Pygame are ready for use.

```
import pygame
pygame.init()
```

3.2 Setting Up the Display

Games fundamentally rely on visuals, requiring a stage or setting for graphics, movements, and player interactions. This 'stage' is typically a window or screen. Within Pygame, we refer to this stage as a 'display surface'.

The display surface is more than just a window; it is the primary drawing area where all game elements are rendered. When you place characters, backgrounds, obstacles, or any graphical component in your game, you're drawing them onto this surface. By setting up this surface, you give your game a defined space with specific dimensions where all visuals will be constrained.

In the Pygame framework, the display surface is created using the `pygame.display.set_mode()` function:

```
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
```

In the code snippet above, a display surface measuring 800x600 pixels is established. This configuration confines our game's visual content to an area 800 pixels wide and 600 pixels high. The display surface's dimensions can be modified to suit the specific needs of your game.

3.3 Setting Up the Game Loop in Pygame

At the heart of nearly every game lies the 'game loop'. It runs continuously during the game, handling events, updating the game state, drawing the game state, and refreshing the display. Below we will explore these steps using a simple example in Pygame: moving a blue rectangle using arrow keys.

Before entering the game loop, we initialize several variables. The variable `running` is set to `True` and is used to control the duration of the game loop. The position of the rectangle (`rect_x` and `rect_y`) is also initialized to define the starting point of the rectangle on the screen.

```
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))
rect_x, rect_y = 100, 100  % Initial position of the rectangle
rect_width, rect_height = 60, 40  % Size of the rectangle
running = True  % Control variable for the game loop
```

3.3.1 Handle any events

The first step in the game loop is handling events, which includes responding to user interactions, such as key presses. In Pygame, events are stored in a queue and can be accessed using `pygame.event.get()`.

```
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False  % Stop the game loop if the window is closed
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                rect_x -= 1  % Move rectangle left
            elif event.key == pygame.K_RIGHT:
                rect_x += 1  % Move rectangle right
            elif event.key == pygame.K_UP:
                rect_y -= 1  % Move rectangle up
            elif event.key == pygame.K_DOWN:
                rect_y += 1  % Move rectangle down
```

The `QUIT` event is triggered when the user closes the game window, typically by clicking the close button on the window. This event sets the `running` variable to `False`, which will terminate the game loop and close the game. The `KEYDOWN` event is used to detect when the user presses an arrow key, and accordingly, the position of the rectangle is updated.

3.3.2 Draw the current game state

After handling the events and updating the game state, the next step is to render the updated elements on the screen. In our case, this involves drawing a blue rectangle at its new position. The rectangle's color is specified using RGB color values, with blue represented by (0, 0, 255).

```
screen.fill((0, 0, 0))  % Clear screen with black color
pygame.draw.rect(screen, (0, 0, 255), pygame.Rect(rect_x, rect_y, rect_width,
    rect_height))  % Draw blue rectangle
```

This code first clears the screen with a black color to remove the previous frame's drawing. Then, it draws a blue rectangle using the updated position values of `rect_x` and `rect_y`.

3.3.3 Refresh the display

Finally, the game loop concludes with refreshing the display, updating the actual screen with the new graphics. This is done using the `pygame.display.update()` function, which makes the newly drawn frame visible.

```
pygame.display.update()
```

This function call is essential as it updates the entire screen surface to the screen, ensuring that the movements and changes made in the game are rendered and visible to the player.

3.3.4 Complete Game Loop Example

Let's put all these steps together to form a complete game loop for our rectangle moving game.

```
import pygame

# Initialize Pygame
pygame.init()
screen = pygame.display.set_mode((800, 600))

# Variables for rectangle position and size
rect_x, rect_y = 100, 100
rect_width, rect_height = 60, 40

# Game loop control variable
running = True

# Game loop
```

```python
while running:
    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                rect_x -= 1  # Move rectangle left
            elif event.key == pygame.K_RIGHT:
                rect_x += 1  # Move rectangle right
            elif event.key == pygame.K_UP:
                rect_y -= 1  # Move rectangle up
            elif event.key == pygame.K_DOWN:
                rect_y += 1  # Move rectangle down

    # Draw the current game state
    screen.fill((0, 0, 0))  # Clear screen with black color
    pygame.draw.rect(screen, (0, 0, 255), pygame.Rect(rect_x, rect_y,
   rect_width, rect_height))

    # Refresh the display
    pygame.display.update()

# Quit Pygame
pygame.quit()
```

4 Understanding Colors

In Pygame, as well as many other graphical libraries, colors are represented using the RGB (Red, Green, Blue) model. Each component of the RGB tuple can range from 0 to 255, indicating the intensity of that color component. Combining these primary colors in various intensities yields a vast spectrum of colors. Below are some common colors and their RGB representations:

- **Red**: Achieved with the RGB tuple (255, 0, 0), where only the red component is maximized.
- **Green**: Achieved by maximizing the green component. RGB tuple: (0, 255, 0).
- **Blue**: Pure blue is denoted by maximizing only the blue component. RGB tuple: (0, 0, 255).
- **White**: A result of full intensity of all three primary colors. RGB tuple: (255, 255, 255).
- **Black**: An absence of color intensity across all components. RGB tuple: (0, 0, 0).
- **Orange**: A mixture of red and green. RGB tuple: (255, 165, 0).
- **Yellow**: Achieved by combining red and green at full intensity. RGB tuple: (255, 255, 0).
- **Cyan**: A combination of green and blue. RGB tuple: (0, 255, 255).
- **Magenta**: Formed by combining red and blue. RGB tuple: (255, 0, 255).

5 Understanding Coordinates in Pygame

In Pygame, the screen or any image is represented as a grid of pixels. The top-left corner of this grid is the origin, denoted by the point $(0, 0)$.

5.1 Coordinate System

The x-coordinate increases as you move to the right and the y-coordinate increases as you move downwards. This might be different from the coordinate system you learned in mathematics, where the y-coordinate increases upwards.

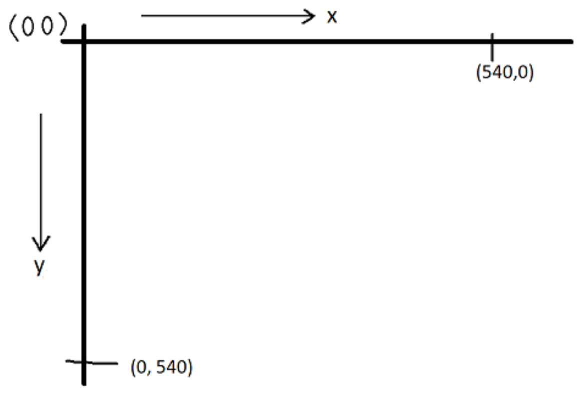


Figure 1: Pygame's coordinate system.

5.2 Positioning Elements

When you're placing an element, say a rectangle or a circle, on the screen, you are essentially specifying where in this grid of pixels the top-left corner of that element should lie.

For example, if you wish to place a rectangle such that its top-left corner is 50 pixels from the left edge of the screen and 100 pixels from the top edge, you would specify its position as $(50, 100)$.

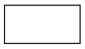
5.3 Width and Height

While the x and y coordinates specify a position, many elements also need a width and height. For instance, when drawing a rectangle, you need to provide not only the top-left corner's position but also how wide and tall the rectangle should be. Now that we understand the basics of positioning and dimensions in the Pygame coordinate system, let's move on to drawing some basic shapes.

6 Drawing Simple Shapes in Pygame

Pygame provides a suite of functions under the ‘pygame.draw‘ module, which can be used to render shapes like lines, rectangles, circles, and polygons directly on the screen surface. Remember, after drawing these shapes, you should update the display using either ‘pygame.display.flip()‘ or ‘pygame.display.update()‘ to make them visible.

6.1 Drawing a Rectangle



To draw a rectangle, use the ‘pygame.draw.rect()‘ function. It requires the screen surface, color, and a Rect object or tuple representing the top-left corner coordinates, width, and height of the rectangle.

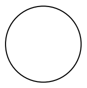
```
red = (255, 0, 0)  # RGB for red color
rect_position = (100, 100, 150, 80)  # (x, y, width, height)
pygame.draw.rect(screen, red, rect_position)
```

To draw an empty rectangle (i.e., just the border) you can provide an additional ‘width‘ parameter:

```
border_width = 2
pygame.draw.rect(screen, red, rect_position, border_width)
```

If the ‘width‘ parameter is omitted or zero, the rectangle will be filled.

6.2 Drawing a Circle



For circles, the ‘pygame.draw.circle()‘ function is used. It requires the screen surface, color, a tuple for the center position, and the radius of the circle.

```
blue = (0, 0, 255)  # RGB for blue color
circle_center = (400, 300)
circle_radius = 50
pygame.draw.circle(screen, blue, circle_center, circle_radius)
```

To draw an empty circle (i.e., just the border) you can provide an additional ‘width‘ parameter:

```
border_width = 2
pygame.draw.circle(screen, blue, circle_center, circle_radius, border_width)
```

If the ‘width‘ parameter is omitted or set to zero, the circle will be filled.

6.3 Drawing a Line

To draw a line, you can use ‘pygame.draw.line()‘. This function requires the screen surface, color, starting point, and ending point.

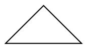
```
green = (0, 255, 0)  # RGB for green color
start_pos = (100, 500)
end_pos = (700, 500)
pygame.draw.line(screen, green, start_pos, end_pos)
```

To adjust the width of the line, you can provide an additional ‘width‘ parameter:

```
line_width = 5
pygame.draw.line(screen, green, start_pos, end_pos, line_width)
```

By default, if the ‘width‘ parameter is omitted, the line will have a width of 1 pixel.

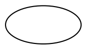
6.4 Drawing Polygons



If you wish to draw polygons, Pygame offers the ‘pygame.draw.polygon()‘ function. It requires a list of at least three coordinate points.

```python
color = (255, 255, 0)  # RGB for yellow
points = [(100, 100), (140, 200), (60, 200)]
pygame.draw.polygon(screen, color, points)
```

6.5 Drawing Ellipses

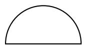


An ellipse can be drawn using the ‘pygame.draw.ellipse()‘ function. This function needs the display surface, color, and a rectangle specification in which the ellipse should fit.

```python
color = (128, 0, 128)  # RGB for purple
ellipse_rectangle = (300, 300, 150, 100)  # x, y, width, height
pygame.draw.ellipse(screen, color, ellipse_rectangle)
```

Note that if the width and height for the rectangle are equal, the ellipse becomes a circle.

6.6 Drawing Arcs



Arcs can be rendered using the ‘pygame.draw.arc()‘ function. This function requires the display surface, color, a rectangle specification to determine the circle from which the arc is taken, and starting and ending angles for the arc.

```python
color = (255, 165, 0)  # RGB for orange
arc_rectangle = (400, 100, 150, 150)  # x, y, width, height
start_angle = 0  # in radians
end_angle = 3.14 / 2  # 90 degrees in radians
pygame.draw.arc(screen, color, arc_rectangle, start_angle, end_angle, 5)  # 5
    is the line width
```

It’s important to note that the angles for the arc function are measured in radians, not degrees.

6.7 Drawing a Point

•

You can draw a point using the ‘pygame.draw.line()‘ function by providing the same start and end coordinates.

```python
color = (255, 255, 255)  # RGB for white
point_position = (50, 50)
pygame.draw.line(screen, color, point_position, point_position)
```

6.8 Complete Pygame Shapes Example

Let us combine everything discussed to showcase a Pygame program that displays the various shapes: Rectangle, Circle, Line, Polygons, Ellipses, Arcs, and a Point. You can use this as a starting point and experiment with positions, sizes, and colors to get more familiar with Pygame's drawing capabilities.

```python
import pygame
import math
pygame.init()
# Setting up the display
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('Pygame Shapes')

# Colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
PURPLE = (128, 0, 128)
ORANGE = (255, 165, 0)

# Main game loop
running = True
while running:
    screen.fill(WHITE)  # Setting a white background

    # Drawing a rectangle
    pygame.draw.rect(screen, RED, (50, 50, 100, 50))
    # Drawing a circle
    pygame.draw.circle(screen, GREEN, (300, 300), 50)
    # Drawing a line
    pygame.draw.line(screen, BLUE, (100, 400), (300, 500), 5)

    # Drawing a polygon (triangle in this case)
    pygame.draw.polygon(screen, BLUE, [(500, 500), (600, 500), (550, 400)])

    # Drawing an ellipse
    pygame.draw.ellipse(screen, PURPLE, (350, 50, 150, 100))

    # Drawing an arc
    pygame.draw.arc(screen, ORANGE, (400, 100, 150, 150), 0, math.pi/2, 5)

    # Drawing a point
    pygame.draw.line(screen, RED, (50, 550), (50, 550))

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    pygame.display.update()

pygame.quit()
```

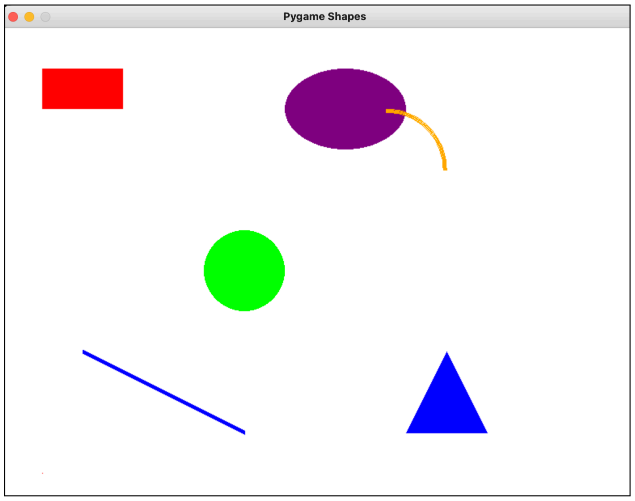


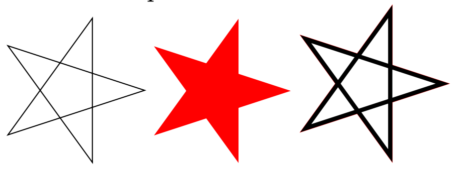
Figure 2: Simple Shapes in Pygame.

6.9 Questions: Creating Shapes in Pygame

Using Pygame, write programs to construct and display the following shapes:

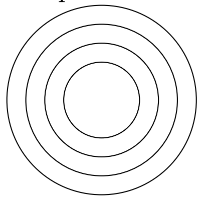
6.9.1 Star:

Draw the following five-pointed stars. This will test your understanding of lines or polygons. Submit your answer to this question in a file named 'stars.py'.



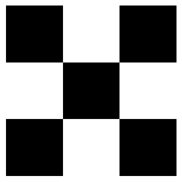
6.9.2 Concentric Circles:

Create 5 circles with the same center point but varying radii, each smaller than the last. Submit your answer to this question in a file named 'circles.py'.



6.9.3 Checkerboard:

Construct an 4x4 checkerboard using squares. To achieve this pattern, consider utilizing nested 'for' loops to alternate between two colors of your choice. Submit your answer to this question in a file named 'board.py'.



Hints and Tips

- Decompose the complex shapes into basic shapes or patterns you know how to draw.
- Use graph paper or drawing software to plan your designs before coding.

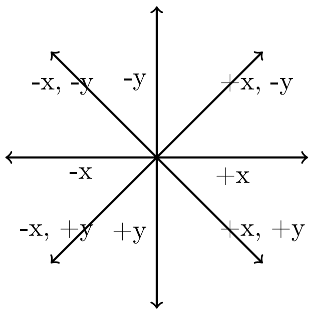
7 Pygame Movement Basics

In Pygame, we make things look like they're moving by quickly drawing them in different spots one after the other. It's like flipping through a book of pictures really fast. This makes it look like the pictures are moving. We use a loop to do this over and over.

7.1 Simple Movements in Different Directions

An object can move in different directions by simply adjusting its x and y coordinates. Here's a brief overview of how this works for eight primary directions:

- **Right (East)**: Increment the x value.
- **Left (West)**: Decrement the x value.
- **Up (North)**: Decrement the y value. (Note: In many graphics systems, the top-left corner is the origin, so decreasing the y-value moves an object up).
- **Down (South)**: Increment the y value.
- **Upper Right (Northeast)**: Increment the x value and decrement the y value.
- **Upper Left (Northwest)**: Decrement both the x and y values.
- **Lower Right (Southeast)**: Increment both the x and y values.
- **Lower Left (Southwest)**: Decrement the x value and increment the y value.



By adjusting the magnitude by which we change x and y in each frame, we can control the speed of the movement in the desired direction.

7.2 Example 1: Movement to the Right

To demonstrate a ball moving from the left side of the screen to the right in Pygame, we follow these simple steps:

1. Initialize the ball's starting position at coordinates (x, y).
2. Increment the x coordinate in each frame to shift the ball rightward.

The following Pygame code snippet illustrates this process. The ball begins on the left side and moves towards the right until it disappears off the screen.

```python
import pygame, sys
from pygame.locals import *

# Initialize pygame
pygame.init()
```

```

# Constants
WIDTH, HEIGHT = 400, 400
SPEED = 2

# Colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)

# Set up the display and clock
DISPLAYSURF = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('Move Ball to the Right')

# Define the ball's initial position
x, y = 10, HEIGHT // 2

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    DISPLAYSURF.fill(WHITE)

    x += SPEED  # Move the ball to the right

    pygame.draw.circle(DISPLAYSURF, RED, (int(x), int(y)), 10)
    pygame.display.flip()
```

The speed of the ball's movement is controlled by the ‘SPEED‘ variable. Once the ball exits the screen, it does not reappear, keeping the example straightforward and focused on linear movement.

7.3 Example 2: Animating a Bouncing Ball

Here's a simplified example illustrating a ball bouncing vertically. The ball's vertical position changes, and when it reaches the top or bottom of the window, it reverses direction to create the bouncing effect.

```
import pygame, sys

# Initialize all imported pygame modules.
pygame.init()

# Set up the display window with a width of 400 and height of 300 pixels.
DISPLAYSURF = pygame.display.set_mode((400, 300), 0, 32)
pygame.display.set_caption('Bouncing Ball Animation')

# Define the color white for later use.
WHITE = (255, 255, 255)

# Set up the ball's properties.
ballRadius = 20
ballX = 200 # Starting horizontal position - center of the window.
ballY = 150 # Starting vertical position - also the center.
speedY = 5  # The number of pixels the ball moves vertically.
```

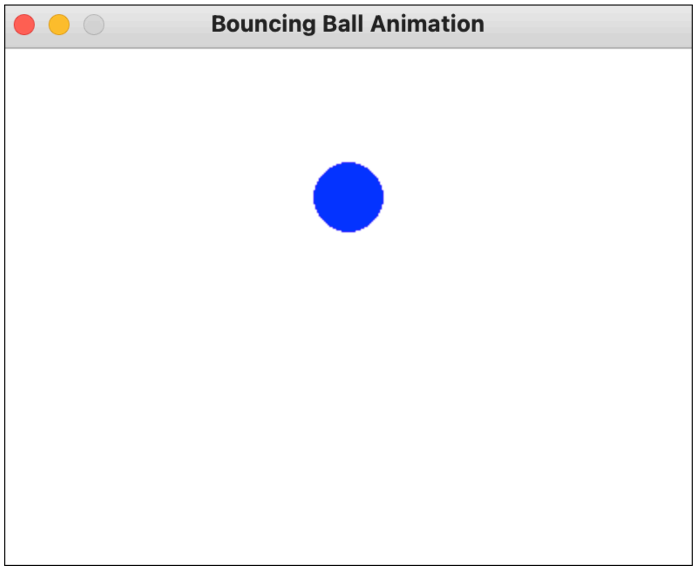


Figure 3: Pygame's coordinate system.

```python
direction = 'down' # Initial direction of ball's movement.

# Main game loop.
while True:
    # Fill the screen with white, effectively erasing any previous drawings.
    DISPLAYSURF.fill(WHITE)

    # Move the ball based on its current direction.
    if direction == 'down':
        ballY += speedY
        # If the ball touches the bottom edge, change its direction to up.
        if ballY >= 300 - ballRadius:
            direction = 'up'
    elif direction == 'up':
        ballY -= speedY
        # If the ball touches the top edge, change its direction to down.
        if ballY <= ballRadius:
            direction = 'down'

    # Draw the ball on the display surface.
    pygame.draw.circle(DISPLAYSURF, (0, 0, 255), (ballX, ballY), ballRadius)

    # Handle events like closing the window.
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()  # Uninitialize all pygame modules.
            sys.exit()     # Exit the program.

    # Update the display to show the latest drawn frame.
    pygame.display.update()
```

7.4 Modified Code with Frame Rate Control

A notable issue with the previous code was the excessively rapid movement of the ball. This occurred because the loop iterated and updated the ball's position as quickly as the system allowed, leading to an uncontrolled speed of movement. This resulted in inconsistent performance across different systems; for instance, a faster system would cause the ball to bounce more quickly, while on a slower system, the movement would appear slower.

To remedy this, we can introduce a frame rate control using Pygame's 'Clock' object. This allows us to regulate the speed at which the game loop iterates, providing a consistent experience regardless of the system's performance capabilities. The frame rate is defined by the 'FPS' (Frames Per Second) variable. The modified code below demonstrates how to implement frame rate control in our ball bouncing example:

```python
import pygame, sys

pygame.init()

FPS = 60 # frames per second setting
# Initialize a clock object to control frame rate
fpsClock = pygame.time.Clock()

# set up the window
DISPLAYSURF = pygame.display.set_mode((400, 300), 0, 32)
pygame.display.set_caption('Bouncing Ball Animation')

WHITE = (255, 255, 255)
ballRadius = 20
ballX = 200 # starting horizontal position
ballY = 150 # starting vertical position
speedY = 5 # speed of ball's vertical movement
direction = 'down' # ball's starting direction

while True:
    DISPLAYSURF.fill(WHITE)

    if direction == 'down':
        ballY += speedY
        if ballY >= 300 - ballRadius: # 300 is window's height
            direction = 'up'
    elif direction == 'up':
        ballY -= speedY
        if ballY <= ballRadius:
            direction = 'down'

   # Drawing the ball on the screen
    pygame.draw.circle(DISPLAYSURF, (0, 0, 255), (ballX, ballY), ballRadius)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    pygame.display.update()
    # Ensure the loop runs only 30 times per second
    fpsClock.tick(FPS)
```

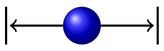
The method ‘fpsClock.tick(FPS)‘ serves to regulate the game’s speed. Technically, the ‘tick‘ method will delay to ensure the game doesn’t run faster than the FPS we specify. If the game loop cycle takes shorter than the time specified by ‘FPS‘, it will pause to meet that frame rate. For instance, with an ‘FPS‘ value of 30, the loop will not iterate more than 30 times in one second. If a loop cycle takes 0.01 seconds (10 milliseconds), then ‘tick(30)‘ will pause for an additional 0.0233 seconds (23.3 milliseconds), making the entire loop iteration last 1/30th of a second.

You may experiment by changing the FPS constant in the program. Lower values will slow down the animation, whereas higher values will speed it up.

Frame Rate in Every Game Loop: Incorporating frame rate control, as highlighted by the three lines marked in red in the previous example, into every game’s main loop is imperative. This ensures consistent gameplay across various devices, guarding against potential issues or glitches stemming from processing speed variations. Without this control, players could face erratic behaviors or varying performance levels based on their system’s capabilities.

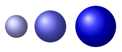
7.5 Questions: Pygame Movement Basics

7.5.1 Horizontal Bouncing Ball:



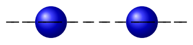
Modify the provided code (from Section 7.4) to enable the ball to bounce horizontally rather than vertically. The ball should move from left to right, rebounding off the window’s sides. Save your script as Horizontal-Bouncing-Ball.py.

7.5.2 Changing Ball Size:



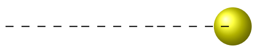
Introduce a feature where the ball enlarges every time it bounces off a window edge. Ensure the ball does not outgrow the window. Once the ball reaches a certain diameter, have it revert to its original size. Save your script as Changing-Ball-Size.py.

7.5.3 Multiple Bouncing Balls:



Introduce an additional ball to the window, ensuring it travels at a different speed. Both balls should bounce off the window’s edges. Save your script as Multiple-Bouncing-Balls.py.

7.5.4 Color Change on Bounce:



Modify the code to change the ball’s color each time it bounces off an edge. The ball should cycle through a list of predefined colors. Save your script as Diff-Color-Bouncing-Ball.py.

8 Different Movement Patterns

Various movement patterns can be implemented using Pygame. Understanding the underlying mathematical concepts allows us to successfully implement these patterns in our game designs.

8.1 Character Movement with Vectors

This section explains how a character or object can seamlessly transition between points. For example, consider a blue circle that needs to move from one point to another on the screen, such as following the mouse cursor.

8.1.1 Direction and Distance in Movement

When moving a character from point (x_1, y_1) to (x_2, y_2), it's essential to determine the direction, and the distance of the movement. While direction indicates the path along which the character moves and distance indicates how far the character travels.

Calculating the Direction: The direction is akin to an arrow pointing from the starting point to the target point. It is calculated by finding the differences in the x and y coordinates:

- Horizontal difference $dx = x_2 - x_1$
- Vertical difference $dy = y_2 - y_1$

For instance, when moving from $(1, 10)$ to $(75, 333)$, we calculate $dx = 74$ and $dy = 323$. This direction vector indicates both the direction and the distance the circle needs to travel to reach the target point, which in this case is the mouse cursor located at $(75, 333)$. The positive values of dx and dy imply that the circle must move rightward and downward to reach the target. Conversely, if $dx = 10$ and $dy = -88$, it would suggest that the circle needs to move rightward and upward to reach its target.

Measuring the Distance: The distance is the length of our direction arrow. We use the Pythagorean theorem to calculate it:

$$\text{distance} = \sqrt{(dx)^2 + (dy)^2}$$

For our example where the blue circle moves from $(1, 10)$ to $(75, 333)$, we have already calculated $dx = 74$ and $dy = 323$. Using these values, the distance the circle needs to travel is calculated as:

$$\text{distance} = \sqrt{74^2 + 323^2} = \sqrt{5476 + 104329} = \sqrt{109805} \approx 331.37$$

This distance represents the total straight-line distance the circle must travel to reach the cursor at $(75, 333)$ from its starting position at $(1, 10)$.

8.1.2 Normalizing the Direction Vector for Consistent Velocity

Normalization is the process of resizing the direction vector to a unit length (making it a unit vector) while maintaining its direction. This crucial step ensures that our character moves at a consistent speed, which is a key component of velocity, regardless of the distance to the target.

In our example of moving the blue circle from $(1, 10)$ to $(75, 333)$, we start with the direction vector components $dx = 74$ and $dy = 323$, and the total distance distance $= 331.37$. To normalize the direction vector, we divide each component by the total distance, obtaining a unit vector that points in the intended direction:

$$\text{Normalized dx} = \frac{dx}{\text{distance}} = \frac{74}{331.37} \approx 0.223$$
$$\text{Normalized dy} = \frac{dy}{\text{distance}} = \frac{323}{331.37} \approx 0.975$$

The resulting normalized direction vector, approximately $(0.223, 0.975)$, has a length of 1 and retains the same direction as the original vector. When this normalized vector is multiplied by a speed factor, it forms the velocity vector, allowing the circle to move towards the target with a uniform speed in each frame. This approach ensures that the velocity of the character is consistent, providing smooth and predictable movement within the game.

8.1.3 Applying the Movement

Continuing with our example, let's say our blue circle starts at $(1, 10)$ and the target position is $(75, 333)$. After normalizing the direction vector to $(0.223, 0.975)$, we apply this movement in the game loop. Consequently, in each frame, the circle moves closer to the destination. For instance, for the first five frames, the circle's position will be updated as follows:

- Frame 1: $(1 + 0.223, 10 + 0.975) = (1.223, 10.975) \rightarrow (1, 11)$.
- Frame 2: $(1.223 + 0.223, 10.975 + 0.975) = (1.446, 11.950) \rightarrow (1, 12)$.
- Frame 3: $(1.446 + 0.223, 11.950 + 0.975) = (1.669, 12.925) \rightarrow (2, 13)$.
- Frame 4: $(1.669 + 0.223, 12.925 + 0.975) = (1.892, 13.900) \rightarrow (2, 14)$.
- Frame 5: $(1.892 + 0.223, 13.900 + 0.975) = (2.115, 14.875) \rightarrow (2, 15)$.

Similarly, in the last five frames before reaching the target position, the circle's position updates as follows:

- $(74.025, 328.742) \rightarrow (74, 329)$.
- $(74.248, 329.717) \rightarrow (74, 330)$.
- $(74.471, 330.691) \rightarrow (74, 331)$.
- $(74.694, 331.666) \rightarrow (75, 332)$.
- $(74.918, 332.641) \rightarrow (75, 333)$.

This example illustrates how the circle smoothly transitions towards the target position, updating its coordinates incrementally in each frame based on the normalized direction vector. To increase the circle's movement speed toward the destination in each frame, you may multiply the velocity by a higher speed factor (e.g., 5). For instance:

- Frame 1: $(1 + (0.223 \times 5), 10 + (0.975 \times 5)) = (2.115, 14.875) \rightarrow (2, 15)$.

8.1.4 Complete Pygame Example

```python
import pygame
import math

# Initialize Pygame
pygame.init()

# Set up the display
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Follow the Mouse")

# Circle settings
circle_color = (0, 0, 255)  # Blue color
circle_radius = 20
circle_position = [400, 300]  # Start position
speed = 2  # Speed of the circle's movement
```

```python

# Main game loop
running = True
while running:
    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Get mouse position
    mouse_x, mouse_y = pygame.mouse.get_pos()

    # Calculate direction towards the mouse
    direction_x = mouse_x - circle_position[0]
    direction_y = mouse_y - circle_position[1]
    distance = math.sqrt(direction_x**2 + direction_y**2)

    # Normalize the direction
    if distance > 0:
        direction_x /= distance
        direction_y /= distance

    # Move the circle towards the mouse
    circle_position[0] += direction_x * speed
    circle_position[1] += direction_y * speed

    # Update the screen
    screen.fill((255, 255, 255))  # Fill the screen with white
    pygame.draw.circle(screen, circle_color, (int(circle_position[0]), int(
        circle_position[1])), circle_radius)
    pygame.display.flip()
    pygame.time.delay(10)

# Quit Pygame
pygame.quit()
```

8.2 Circular Movement

Circular movement can be thought of as an object moving around the circumference of a circle. The movement's speed can be adjusted by controlling the angle of movement for each frame.

Given a circle with a center (h, k) and a radius r, an object can move around this circle using the parametric equations:

$$x = h + r\cos(\theta)$$
$$y = k + r\sin(\theta)$$

Where θ is the angle of movement. By adjusting θ for each frame, we can control the object's speed around the circle.

Example: Circular Movement

To demonstrate the concept of circular movement, let's create a simple Pygame script that moves a ball around the circumference of a circle.

```python
import pygame, sys, math

```

```python
# Initialize pygame
pygame.init()

# Constants
WIDTH, HEIGHT = 400, 400
CENTERX, CENTERY = WIDTH // 2, HEIGHT // 2
RADIUS = 100

# Colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)

# Initialize the display and clock
DISPLAYSURF = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('Circular Movement')
fpsClock = pygame.time.Clock()
angle = 0

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    DISPLAYSURF.fill(WHITE)

    # Calculate the x and y coordinates based on the angle
    x = CENTERX + RADIUS * math.cos(math.radians(angle))
    y = CENTERY + RADIUS * math.sin(math.radians(angle))

    pygame.draw.circle(DISPLAYSURF, RED, (int(x), int(y)), 10)

    # Update the display
    pygame.display.flip()

    # Increment the angle
    angle += 1  # Increase this value to make the ball move faster.
    if angle >= 360:
        angle = 0

    fpsClock.tick(60)
```

In the above code, a ball moves around a circle of radius ‘RADIUS‘ centered at ‘(CENTERX, CENTERY)‘. The angle is incremented by one degree in each frame, causing the ball to make a complete revolution around the circle in 360 frames.

8.3 Zigzag Movement Pattern

A zigzag movement pattern means moving in a back-and-forth manner, similar to how a ball bounces off walls or how a lightning bolt travels through the sky. It’s like drawing a series of connected diagonal lines where the direction changes sharply at each point. This pattern is easy to spot when an object, such as a ball in a game, moves across the screen or field in this distinct way, creating a zigzag path as it goes.

For a downward zigzag pattern:

1. The object starts at an initial position (x, y).

2. It moves in a diagonal direction (either down-right or down-left) for a predetermined span.
3. Once this span is covered, the horizontal direction is switched, while the downward motion continues consistently.
4. The process repeats, creating a series of diagonal segments that give the impression of a zigzag.

This pattern is particularly useful in gaming scenarios where unpredictable or evasive movements are required.

Example: Downward Zigzag Pattern Movement

In this Pygame example, a ball moves in a downward zigzag pattern, shifting its horizontal direction after covering a consistent horizontal span:

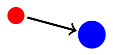
```python
import pygame, sys

# Initialize pygame
pygame.init()

# Constants
WIDTH, HEIGHT = 400, 400
SPEED = 2
ZIGZAG_SPAN = 40  # Horizontal distance before changing direction

# Colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)

# Initialize the display and clock
DISPLAYSURF = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('Downward Zigzag Pattern Movement')
fpsClock = pygame.time.Clock()

# Ball's starting position and direction
x, y = WIDTH // 2, 0
dir_x = 1
initial_x = x

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    DISPLAYSURF.fill(WHITE)

    x += SPEED * dir_x
    y += SPEED

    # Check if the horizontal zigzag span is covered
    if abs(x - initial_x) >= ZIGZAG_SPAN:
        dir_x *= -1  # Reverse horizontal direction
        initial_x = x  # Reset the initial position for the next span

    pygame.draw.circle(DISPLAYSURF, RED, (int(x), int(y)), 10)
    pygame.display.flip()
    fpsClock.tick(60)
```

This ball starts at the top center of the screen and moves downwards in the depicted zigzag pattern. The horizontal direction switches after each ‘ZIGZAG_SPAN‘, while the vertical (downward) movement remains constant. The ‘SPEED‘ and ‘ZIGZAG_SPAN‘ constants can be adjusted to modify the movement’s speed and span, respectively.

8.4 Questions: Pygame Movement Patters (optional, 20 extra points)

8.4.1 Direct Line Motion

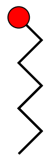
(Extra Credit: 10 points)



Randomly position a circle on the screen and program an object, such as another circle, an image, or a rectangle, to move in a straight line from the bottom left corner towards the circle. Save your script as line-motion.py.

8.4.2 Zigzag Movement Pattern

(Extra Credit: 10 points)



Modify the zigzag pattern such that the object zigzags from the top to the bottom, and then back to the top, repeatedly. Save your script as Zigzags.py.

9 Detecting Events

Pygame offers comprehensive tools for detecting player interactions. Events are actions recognized by Pygame, such as a button press or a mouse movement. Pygame records these events in an event queue, which developers can process and respond to accordingly.

9.1 Handling Keyboard Inputs

Pygame provides the capability to not only detect key presses but also identify which specific key is pressed. To track any key press, the `KEYDOWN` event comes into play:

```python
if event.type == pygame.KEYDOWN:
    # A key has been pressed
```

To cater to scenarios requiring specific key identification, Pygame can distinguish individual key presses:

```python
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_LEFT:
        # Detects the left arrow key press
    if event.key == pygame.K_RIGHT:
        # Detects the right arrow key press
```

9.2 Detecting Mouse Events in Pygame

Beyond the keyboard, Pygame supports detecting various mouse events, from movements to clicks.

To detect mouse movement:

```python
if event.type == pygame.MOUSEMOTION:
    x, y = event.pos
    # x and y hold the new mouse position
```

Pygame can also recognize both mouse button presses and releases:

```python
if event.type == pygame.MOUSEBUTTONDOWN:
    # A mouse button has been pressed
    if event.button == 1:
        # Left mouse button
    if event.button == 3:
        # Right mouse button
```

9.3 Example: Keyboard-Controlled Object Movement

To illustrate how keyboard events work, let's look at a simple scenario: a rectangle moving left or right on the screen when arrow keys are pressed. In the following example, hitting the left arrow key shifts the rectangle left by "rect_speed" pixels, and the right arrow key moves it right.

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Moving Rectangle Example')

# Define the rectangle properties
rect_x = 400
rect_y = 300
rect_width = 50
rect_height = 30
rect_color = (0, 128, 255)
rect_speed = 5

# Create a clock object to control the frame rate
clock = pygame.time.Clock()

running = True
while running:
    screen.fill((255, 255, 255))
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        # Detecting left and right arrow key presses
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                rect_x -= rect_speed
            if event.key == pygame.K_RIGHT:
                rect_x += rect_speed

    pygame.draw.rect(screen, rect_color, (rect_x, rect_y, rect_width,
    rect_height))
    pygame.display.update()

    # Limit the loop to 60 frames per second
    clock.tick(60)
```

The figure below demonstrates how x and y values change based on the direction of movement:

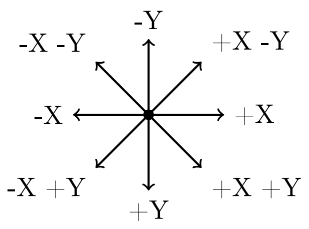


Figure 4: Changes in x and y values based on directional movement.

9.4 Continuous Movement with Arrow Key Presses

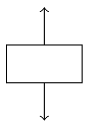
In the previous example, the rectangle moved only once for each arrow key press. To allow continuous movement while an arrow key is pressed and held down, you can maintain the state of the arrow keys. This can be achieved by using boolean flags.

When an arrow key is pressed, its corresponding flag is set to `True`, and when it's released, the flag is set to `False`. During the game loop, you can check these flags and move the rectangle accordingly. Here's how you can implement this:

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Continuously Moving Rectangle Example')

# Define the rectangle properties
rect_x = 400
rect_y = 300
rect_width = 50
rect_height = 30
rect_color = (0, 128, 255)
rect_speed = 5

# Flags for arrow keys
left_pressed = False
right_pressed = False

# Create a clock object to control the frame rate
clock = pygame.time.Clock()

running = True
while running:
    screen.fill((255, 255, 255))

    # Check key states and move rectangle
    if left_pressed:
        rect_x -= rect_speed
    if right_pressed:
        rect_x += rect_speed

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        # Detecting left and right arrow key presses
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                left_pressed = True
            if event.key == pygame.K_RIGHT:
                right_pressed = True
        # Detecting left and right arrow key releases
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT:
                left_pressed = False
            if event.key == pygame.K_RIGHT:
                right_pressed = False

    pygame.draw.rect(screen, rect_color, (rect_x, rect_y, rect_width,
   rect_height))
    pygame.display.update()

    # Limit the loop to 60 frames per second
    clock.tick(60)
```

9.5 Questions: Detecting Events

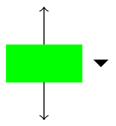
9.5.1 Vertical Movement:



Adjust the example in Section 9.4 to enable vertical movement, utilizing the UP and DOWN arrow keys. The rectangle should move upward with the UP key and downward with the DOWN key.

Hint: You will need to introduce two new boolean flags for the UP and DOWN arrow keys. Save your script as *Up-and-Down-Movement.py*.

9.5.2 Change Rectangle Color on Mouse Click

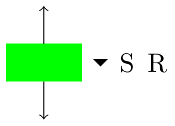


Modify the previous example so that when the left mouse button is clicked anywhere on the game window, the color of the rectangle changes to a random color.

Hint: You can use the `pygame.MOUSEBUTTONDOWN` event and the `random` module to generate random RGB values for the rectangle color.

Save your script as *Change-Rectangle-Color-on-Mouse-Click.py*.

9.5.3 Increase Speed on Key Press



Introduce a new feature to the movement example: when the player presses the 'S' key, the speed of the rectangle should double, and when the 'R' key is pressed, it should return to the original speed.

Hint: Introduce a new event check for the 'S' and 'R' keys and adjust the `rect_speed` variable accordingly.

Save your script as *Increase-Speed-on-Key-Press.py*.

10 Using Images in Pygame

10.1 Loading Images

In Pygame, the `pygame.image.load()` function allows for image loading. The image should be in a Pygame-compatible format like PNG or JPEG. The image file, such as 'image.png', should either be in the same folder as your Python script or you should specify its complete file path.

```python
import pygame
image = pygame.image.load('image.png')
```

10.2 Displaying Images

Once you've loaded an image, you can display it on the screen using the `blit()` method.

10.3 Displaying an Image at a Specific Position

The following code snippet loads and displays an image at coordinates `(x, y)` on the screen.

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))

image = pygame.image.load('image.png')
x = 100
y = 50

screen.blit(image, (x, y))
pygame.display.update()
```

10.4 Example: Moving an Image with Arrow Keys

You can also move an image using keyboard inputs. The following example moves an image (cat) using the arrow keys.

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))
image = pygame.image.load('cat.png')
x, y = 400, 300

# Create a clock object to control the frame rate
clock = pygame.time.Clock()

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        x -= 5
    if keys[pygame.K_RIGHT]:
        x += 5
    if keys[pygame.K_UP]:
        y -= 5
    if keys[pygame.K_DOWN]:
        y += 5

    # Clear the screen
    screen.fill((255, 255, 255))

    screen.blit(image, (x, y))
    pygame.display.update()

    # Limit the loop to 60 frames per second
    clock.tick(60)
```

11 Using Text in Pygame

Pygame provides a way to render text onto the screen using its font module. To display text in Pygame, you'll typically follow these steps:

1. Initialize the font with `pygame.font.init()`.
2. Create a font object using `pygame.font.Font()`.
3. Render the text using the `render()` method of the font object.
4. Display the rendered text onto the screen using `blit()`.

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))

# Initialize the font module
pygame.font.init()

# Create a font object using 'freesansbold'.
# freesansbold is one of the default fonts that come bundled with Pygame.
# Typically, the freesansbold.ttf file is located within the Pygame library's
    files.
font = pygame.font.SysFont('freesansbold', 36)

# Render the text. "True" means anti-aliased text.
# (R, G, B) is the color of the text.
text = font.render("Hello, Pygame!", True, (255, 255, 255))

# Position where the text will be displayed
text_position = (400 - text.get_width() // 2, 300 - text.get_height() // 2)

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))
    screen.blit(text, text_position)
    pygame.display.update()
```

You can also change the font, size, color, and other properties of the text to better fit your game's aesthetics. Exploring the `pygame.font` module documentation can provide a deeper understanding of the available functionalities.

12 Playing Sounds in Pygame

In Pygame, you can add audio effects easily to make your games more interactive and lively. The library provides functionalities for both sound effects (like short clips) and background music.

12.1 Sound Effects

Sound effects are typically short clips that you might want to play in response to game events. For example, when a player collects a coin or jumps, you might play a brief sound. To play sound effects, Pygame uses the `pygame.mixer.Sound` class:

```python
import pygame

pygame.init()
pygame.mixer.init()  # Initialize the mixer module

# Load a sound
coin_sound = pygame.mixer.Sound('coin.wav')

# Play the sound
coin_sound.play()
```

You can control the volume, number of repeats, and more with additional methods provided by the `Sound` class.

12.2 Background Music

For longer music tracks, such as background music that should loop throughout gameplay, you would use the `pygame.mixer.music` module:

```python
import pygame

pygame.init()
pygame.mixer.init()

# Load music
pygame.mixer.music.load('background.mp3')

# Play the music infinitely (until stopped)
pygame.mixer.music.play(-1)

# ... Your game loop, events, etc.

# To stop the music
# pygame.mixer.music.stop()
```

Remember, the format of your audio file (like WAV, MP3, etc.) should be supported by the SDL mixer, which Pygame uses. If you encounter issues, you might need to convert your audio files to a different format.

13 Pygame Sound Example

In the Pygame example below, background music continuously plays during the program's execution. Additionally, every time the player clicks on the screen, a sound effect (representative of a coin collection) is triggered. Ensure that the sound files 'coin_effect.wav' and 'game_background_music.mp3' are placed in the same directory as this script for it to function correctly.

```python
import pygame
import sys

pygame.init()
pygame.mixer.init()  # Initialize the mixer module

# Screen setup
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Sound Example')

# Load sound effect and background music
coin_sound = pygame.mixer.Sound('coin_effect.wav')
pygame.mixer.music.load('game_background_music.mp3')

# Play background music on loop
pygame.mixer.music.play(-1)

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        # Play sound effect on mouse click
        if event.type == pygame.MOUSEBUTTONDOWN:
            coin_sound.play()

    screen.fill((0, 0, 0))
    pygame.display.flip()

pygame.mixer.music.stop()  # Stop background music
pygame.quit()
sys.exit()
```

14 Questions: Using Text and Sound

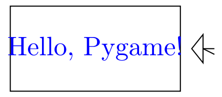
14.1 Image Manipulation:

Extend the example from Section 10.4. Incorporate the following enhancements:

- Rotate the image using the 'R' key. You should use the `pygame.transform.rotate()` function to adjust the image's orientation by a specified angle.
- Increase or decrease the image size with the '+' and '-' keys, respectively. Use the `pygame.transform.scale()` function to modify the image dimensions based on a given scale factor.

Save your script as `Image-Manipulation.py`.

14.2 Text Animation and Sound Effects:



Refer to the text rendering example in Section 11. Make the text "Hello, Pygame!" move and bounce off the screen edges. Augment the animation with these features:

- *Color Change:* Upon hitting an edge, the text should change to a random color.
- *Sound Effect:* Trigger a sound effect every time the text collides with an edge.
- *Background Music:* Integrate continuous background music that plays for the entire duration of the program.

Save your script as `Text-Animation-and-Sound-Effects.py`.

15 Detecting Collision

Collision detection checks if game objects overlap or ‘collide.’ Pygame offers methods for this using bounding rectangles. These rectangles wrap around an object (e.g., image), approximating its shape. This makes it easy to detect overlaps without checking each pixel, ensuring a faster and efficient collision detection.

15.1 Rectangle Collision

The most basic collision detection in Pygame is between two rectangles. Pygame’s `Rect` object has a method called `colliderect()` which can be used to determine if two rectangles overlap.

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))

# Create two rectangles
rect1 = pygame.Rect(350, 250, 100, 100)
rect2 = pygame.Rect(400, 300, 100, 100)

# Drawing rectangles on the screen
pygame.draw.rect(screen, (255,0,0), rect1)
pygame.draw.rect(screen, (0,0,255), rect2)

# Refresh display
pygame.display.flip()

# Check for collision
if rect1.colliderect(rect2):
    print("The rectangles collide!")

# Event loop to keep the window open
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

pygame.quit()
```

15.2 Image Collision Detection

While images have diverse shapes and sizes, detecting collisions pixel-by-pixel is computationally intensive. Thus, we often convert an image into a bounding rectangle for easier collision checks. When an image is loaded in Pygame, it comes with a `get_rect()` method that returns its bounding rectangle.

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Image Collision Detection")

screen.fill((255, 255, 255))  # Fill screen with white

# Load two images
```

```python
image1 = pygame.image.load('cat.png')
image2 = pygame.image.load('cat2.png')

# Get their rectangles
rect1 = image1.get_rect(topleft=(100,100))
rect2 = image2.get_rect(topleft=(150,150))

# Draw the images on the screen
screen.blit(image1, rect1.topleft)
screen.blit(image2, rect2.topleft)

# Check for collision
if rect1.colliderect(rect2):
    pygame.draw.rect(screen, (255, 0, 0), rect1, 2)  # Draw a red border
   around image1 if it collides
    pygame.draw.rect(screen, (255, 0, 0), rect2, 2)  # Draw a red border
   around image2 if it collides
    print("The images collide!")

pygame.display.flip()

# Main loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

pygame.quit()
exit()
```

15.3 Questions: Interactive Rectangle Collision Detection

You'll need to implement a simple collision detection mechanism.

Instructions:

1. Initialize a Pygame window of size 800×600.
2. Create a static rectangle positioned in the center of the screen.
3. Generate another rectangle that's placed at a random location within the window.
4. Allow the user to move this second rectangle using the arrow keys: up, down, left, and right.
5. If the two rectangles collide, play a beep sound to alert the user.
6. Ensure that the moving rectangle cannot go outside the screen boundaries.

Requirements:

- Submit your Pygame code in a file named `moving-rect-collide.py`.
- Also, submit the beep sound file you used in your program.