Chapter 11: Model Solutions

Below, you'll find sample solutions to the lab exercises in the book.

Lab Exercises 11.1

1. A surface in Pygame represents a rectangular area for drawing and manipulating graphics. It acts as a canvas that can be filled with colors, images, or other graphical elements. Surfaces are used to create and display visuals, handle input events, and perform transformations like scaling, rotating, or flipping.

2. To scale a surface, you can use the pygame.transform.scale() function.

   To rotate a surface, you can use the pygame.transform.rotate() function.

   To flip a surface horizontally or vertically, you can use the pygame.transform.flip() function.

3. To ensure smooth gameplay and optimal performance in Pygame, consider the following strategies:

   - Use double buffering by creating a separate surface for drawing and then blitting it to the screen surface.
   - Limit the number of surfaces that need to be updated each frame.
   - Use hardware acceleration if available.
   - Avoid unnecessary surface conversions or transformations.
   - Profile your code and optimize performance-critical sections.

4. A Pygame program that displays a window with a given width and height and fill the window with a specific background color.

```
import pygame

pygame.init()

width, height = 800, 600

window = pygame.display.set_mode((width, height))

pygame.display.set_caption("My Game")

background_color = (255, 0, 255)

window.fill(background_color)

pygame.display.flip()

running = True

while running:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False
```

```
pygame.quit()
```

5. A program that displays a shape in the center of the window.

```
import pygame

pygame.init()

width, height = 800, 600

window = pygame.display.set_mode((width, height))

pygame.display.set_caption("My Game")

background_color = (255, 255, 255)  #white

shape_color = (255, 0, 0)           # red

shape_width = 100

shape_height = 100

shape_x = width // 2 - shape_width // 2

shape_y = height // 2 - shape_height // 2

window.fill(background_color)

pygame.draw.rect(window, shape_color, (shape_x, shape_y,
shape_width, shape_height))

pygame.display.flip()

running = True

while running:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False

pygame.quit()
```

6. Implement keyboard input for exercise 5.

```
import pygame

pygame.init()

width, height = 800, 600

window = pygame.display.set_mode((width, height))

pygame.display.set_caption("My Game")

background_color = (255, 255, 255)  # white

shape_color = (255, 0, 0)  # red

shape_width = 100

shape_height = 100
```

```
shape_x = width // 2 - shape_width // 2
shape_y = height // 2 - shape_height // 2
movement_x = 0
movement_y = 0
movement_speed = 5
clock = pygame.time.Clock()
shape_rect = pygame.Rect(shape_x, shape_y, shape_width,
shape_height)
window.fill(background_color)
pygame.draw.rect(window, shape_color, shape_rect)
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                movement_x = -movement_speed
            elif event.key == pygame.K_RIGHT:
                movement_x = movement_speed
            elif event.key == pygame.K_UP:
                movement_y = -movement_speed
            elif event.key == pygame.K_DOWN:
                movement_y = movement_speed
        else:
            movement_x = 0
            movement_y = 0
    shape_rect.x += movement_x
    shape_rect.y += movement_y
    window.fill(background_color)
    pygame.draw.rect(window, shape_color, shape_rect)
    pygame.display.flip()
```

```
        clock.tick(60)  # Adjust the value to control frame rate
    pygame.quit()
```

7. Sprites in Pygame are visual game objects that can be used to represent characters, enemies, items, or any other element within a game. They are often used to create animated objects by cycling through a series of images or frames. Pygame provides the pygame.sprite.Sprite class as a base class for creating sprites. By subclassing this class and adding custom behavior and attributes, you can create and animate game objects.
8. Use the pygame.image.load() function to load an image file from disk.

```
import pygame

pygame.init()

width, height = 800, 600

window = pygame.display.set_mode((width, height))

pygame.display.set_caption("Image Loading Example")

image = pygame.image.load("test.jpg")

image_width, image_height = image.get_width(),
image.get_height()

image_x = width // 2 - image_width // 2

image_y = height // 2 - image_height // 2

window.fill((255, 255, 255))  # white

window.blit(image, (image_x, image_y))

pygame.display.flip()

running = True

while running:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False

pygame.quit()
```

9. Use the colliderect() method. The rectangle turns blue when a collision is detected.

```
import pygame

# Initialize Pygame

pygame.init()

# Set window width and height

width, height = 800, 600
```

```
# Create the window surface
window = pygame.display.set_mode((width, height))
pygame.display.set_caption("Collision Detection Example")
# Set up rectangles
rect1 = pygame.Rect(100, 100, 100, 100)
rect2 = pygame.Rect(300, 300, 100, 100)
# Set colors
color1 = (255, 0, 0)  # red
color2 = (0, 255, 0)  # green
# Fill the window with a background color
window.fill((255, 255, 255))  # white
# Create a clock object to control the frame rate
clock = pygame.time.Clock()
# Game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # Fill the window with a background color
    window.fill((255, 255, 255))  # white
    # Move the rectangles
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        rect1.x -= 5
    if keys[pygame.K_RIGHT]:
        rect1.x += 5
    if keys[pygame.K_UP]:
        rect1.y -= 5
    if keys[pygame.K_DOWN]:
        rect1.y += 5
    # Check for collision between rect1 and rect2
```

```
    if rect1.colliderect(rect2):
        color1 = (0, 0, 255)  # blue
    else:
        color1 = (255, 0, 0)  # red
    # Draw the rectangles
    pygame.draw.rect(window, color1, rect1)
    pygame.draw.rect(window, color2, rect2)
    # Update the display
    pygame.display.flip()
    # Set the frame rate
    clock.tick(60)  # 60 frames per second
# Quit Pygame
pygame.quit()
```