Snake Game

Aaron Connolly

Introduction/Explanation

This was created using the programming language Python with Pycharm using the pyjama extension. The user is prompted to press any button and the game will start. The user can move around the snake with the four arrow keys. The aim of the game is to eat the fruit which in turn will make the snake grow by one square and increases the score by 1. When the snake fills the entire map, then the user wins.
If the snake crashes its head into its own body or a boundary wall then they will die and a message will appear on the screen displaying their score.

Game Constants and variables:
These constants configure the game's grid layout and dimensions. They also make the code more manageable by keeping all key settings in one place for easy adjustment.

Game Initialisation:
pygame.init() initializes all the pygame modules. pygame.display.set_mode() creates a window of specified size, and pygame.time.Clock() manages the game's frame rate.

Snake and Fruit Initialisation:
The snake starts with three segments. fruitSpawn() places the fruit randomly while ensuring it doesn't overlap with the snake's body.

Direction & Game State Variables:
direction controls the movement of the snake, and game_started ensures the game only begins when an arrow key is pressed.

Event Handling (Key Press):
pygame.event.get() captures player events (like key presses) and updates the direction of the snake, ensuring it moves when the player interacts with the game.

Snake movement Logic:
The position of the snake's head changes depending on which key is pressed. The blockSize ensures that the snake moves by one grid unit.

Snake Growth & Fruit Collision:
When the head of the snake reaches the fruit's position, the body grows, and the game proceeds to place a new fruit elsewhere.

Drawing the Background:

Creating the Background Surface:
The first line creates a background surface that matches the size of the game window using pygame.Surface(game_window.get_size()).

Setting Grid Dimensions and Colours:
The variables w and h represent the width and height of the grid, factoring in the offsets to ensure that the game area is well-centred. c1 and c2 are two shades of green used to create the checkered effect.

Generating the Grid:
The line:
```
tiles = [((x * blockSize + offset_x, y * blockSize +
offset_y, blockSize, blockSize), c1 if (x + y) % 2 == 0 else
c2)
          for x in range((w + blockSize - 1) // blockSize)
          for y in range((h + blockSize - 1) // blockSize)]
```
generates a list of rectangles (tiles) that form the background grid. Each rectangle represents one "tile" in the checkered pattern, with its size determined by blockSize. The colour alternates between c1 and c2 based on the sum of the x and y indices, creating a checkerboard look.

Drawing the Grid:
The line:
```
[pygame.draw.rect(background, color, rect) for rect, color in
tiles]
```
loops through the list of tiles and uses pygame.draw.rect() to draw each rectangle in the specified colour on the background surface.

Displaying the Background:
Finally, game_window.blit(background, (0, 0)) displays the entire background grid on the main game window at position (0, 0).

Drawing the Snake & Fruit:
The snake's body is drawn using a loop, with the head being a different colour to make it visually distinct from the rest of the body.

Collision Detection & Game Over:
These conditions ensure the game ends when the snake hits the boundaries or runs into its own body, this will display the losing message. If the player wins then victory will be equal to True and the victory message will be displayed.

Score Display & Game over message:
The score is rendered using the pygame.font.SysFont() and blitted onto the game window, allowing real-time score updates.

Refreshing The Game Window:
pygame.display.update() refreshes the window so the player can see the updated state, and fps.tick(snake_speed) controls how fast the game runs.