## Code Explanation:

1. The code starts by importing the necessary libraries.
2. These are pygame, time, and random.
3. Next, the code defines some variables.
4. The snake_speed variable controls how fast the snake moves around the screen.
5. The window_x and window_y variables define the size of the game window onscreen.
6. The next line of code initializes pygame.
7. This is important because it sets up all of the game objects and their properties so that they can be used later in the program.
8. Next, the code creates an instance of pygame's GameWindow class object.
9. This object represents a rectangular area onscreen that can be filled with graphics and text content.
10. The GameWindow object has two properties: width and height .
11. These values represent how wide and tall the game window is respectively.
12. The next line of code assigns values to these properties based on a user-defined value called snake_speed .
13. This variable tellspygame how fast (in pixels per second) to move the snake aroundthe screen.
14. Higher values will make for faster movement but also more intense gameplay!
15. Next, PyGame starts loading various images into memory to use as background graphics for our game world .
16. First it loads in an
17. The code will create a window with dimensions of 720×480 pixels.
18. The colours black, white, red, green and blue will be used to represent the game's various elements.
19. Next, the pygame module will be imported and initialized.
20. This will allow us to start working with the game's various objects and functions.
21. The game's main loop will then be started by calling pygame.init().
22. This function will ensure that all of the necessary modules are loaded and ready for use.
23. Finally, we'll call the window's constructor to create our game window.
24. The code starts by creating a pygame.display.set_mode() function to set the window size and position.
25. The code then creates a game window and sets its mode to (0, 0).
26. Next, the code defines some variables: fps, snake_position, snake_body, and fruit_position.

27. These variables will be used to control the speed of the snake, where it starts from (snake_position), how wide it is (snake_body), where the fruit is located (fruit_position), and whether or not fruit should spawn (fruit_spawn).
28. The next block of code calculates the distance between each point on the screen using pygame.time.Clock().
29. This allows us to move the snake around on-screen without having to constantly recalculate its position.
30. Finally, we set up two boolean variables: fruit_spawn and analyze().
31. These will determine whether or not fruit will spawn at random locations on-screen and be analyzed for player input.
32. The code sets up a basic game window with a snake positioned at (100, 50) on the X-axis and (window_x, window_y) on the Y-axis.
33. The FPS controller is initialized and set to run at 60 frames per second.
34. The next block of code defines the body of the snake.
35. A list of ten [100, 50] points is created, starting at position (100, 50).
36. The first four points are set to be in the center of the snake's body while the remaining six points are evenly spaced around it.
37. Next, a fruit position is defined as [(random.randrange(1, (window_x//10)) * 10), (random.randrange(1
38. The code starts by initializing some variables.
39. The first is the score, which starts at 0.
40. The second is the direction variable, which will determine how the snake moves.
41. The show_score() function is called whenever a player makes a choice.
42. This function contains three parts: creating a font object, creating a display surface object, and displaying text on the display surface.
43. First, the score_font object is created.
44. This object stores information about the font used to display text on the screen (in this case, Times New Roman).
45. Next, the score_surface object is created and initialized with information about the font and size of text that will be displayed (50 points in size).
46. Finally, using blit(), the score_rect object is copied onto the score_surface object so that it can be displayed onscreen.
47. The game_over() function ends any current game play and terminates Python code running in this module (assuming no other functions call it).
48. First, an instance of SysFont named my_font is created.

49. Then 50 points in size for Times New Roman are specified as its typeface and color values.
50. Finally, game over() is called to end all game play and terminate Python code running in
51. The code first initializes some variables, including the score variable.
52. The code then creates a function called show_score().
53. This function will be used to display the current score on the screen.
54. The show_score() function first creates a font object called score_font and sets its size to 50 points.
55. Next, the function creates a display surface object called score_surface and sets its color to white.
56. Finally, the show_score() function blits the score_surface object onto the game window's screen.
57. The game over() function is responsible for cleaning up resources after the game has ended.
58. First, it creates a font object called my_font and sets its size to 20 points.
59. Then, the game over() function bl
60. The code first creates a text surface object called game_over_surface.
61. The text will be rendered in the font my_font and the color red.
62. Next, a rectangular object is created for the text surface object.
63. This object will have its midpoint at (window_x/2, window_y/4).
64. Finally, position of the text on the rectangle is set using game_over_rect.midtop().
65. The code creates a text surface object called game_over_surface.
66. This object will be used to display the player's score and the message "Your Score is :".
67. Next, a rectangular object called game_over_rect is created.
68. This object will be used to position the text on the surface.
69. The midpoint of the rectangle is set to (window_x/2, window_y/4).
70. The code starts by initializing the pygame library.
71. Next, the code creates a window and assigns it to game_window.
72. The window has a surface (a graphic representation of the screen) and a Rectangle object that specifies its size and position.
73. Next, the code blits (transfers) the text "GAME OVER" onto the game_over_surface object.
74. The text is drawn in white, centered on top of the game_over_rect object.
75. The program then sets up a timer that will run for 2 seconds.
76. At this point, the program will quit because there is no more code to execute.
77. The code will check for key events and if the event corresponds to a valid key, it will change the text displayed on screen accordingly.

78. If you press any other key, the program will continue to run as normal.
79. The code starts by checking to see if the player has pressed two keys at the same time.
80. If they have, the code changes the direction of the snake.
81. Next, the code checks to see if either key was pressed in a different direction than expected.
82. If it was, then the code adjusts the position of the snake accordingly.
83. Finally, it updates how big the snake's body is getting.
84. The code will check if the two keys being pressed at the same time are either 'UP' or 'DOWN'.
85. If they are, then the direction of the snake will be changed accordingly.
86. If the two keys being pressed are not equal, then the code will check to see if they are different directions.
87. If they are not, then the snake's position will be adjusted by 10 pixels in each direction.
88. Lastly, a function is created that will change how big the snake's body grows when it moves.
89. The code starts by creating a list of snake positions.
90. The first position in the list is at (0, 0), and the last position in the list is at (window_x-10, window_y-10).
91. Next, the code checks to see if any of the positions in the snake are equal to a fruit position.
92. If so, then that fruit gets scored 10 points and is added to the fruit spawn variable.
93. If no fruits are found, then the game moves on to checking for collisions between snakes and fruits.
94. If two snakes intersect, then their scores are incremented by 10.
95. If a snake collides with a wall or another snake, then that snake dies and game over conditions are triggered.
96. Finally, touching any part of a snake causes it to die and also triggers game over conditions.
97. The code will check to see if two positions in the snake body are equal.
98. If they are, then the score is incremented by 10 and the game_over() function is called.
99. If a player touches the snake body at any point, then the game_over() function will be called.