

**Snake Game Project in Python Using Pygame**

The Snake game is a classic arcade game that you can find on many early mobile phones and computer systems. The goal of this project is to recreate the Snake game using Python and the Pygame module. Below, you will find a detailed guide to understanding the code along with a separate, complete code snippet.

**Project Overview**

In the Snake game, the player controls a snake that moves around the screen. The objective is to eat food items that appear randomly on the screen. Each time the snake eats a food item, it grows longer, and the player earns points. The game ends if the snake collides with the screen boundaries or itself.

**Key Components**

**1. Game Window**

The game window is where the snake will move around and where the food items will appear. You can customize the size of the window and the colors used for the snake, food, and background.

**2. Snake**

The snake is represented as a series of blocks. It starts with a certain length and grows longer each time it eats the food. The snake's movement is controlled using the arrow keys.

**3. Food**

The food appears randomly on the screen. When the snake eats the food, a new piece of food appears at a different location.

**4. Game Logic**

The game logic includes collision detection (with the walls and itself), score tracking, and game over conditions.

**Detailed Code Explanation**

Below is the complete Python code for the Snake game using the Pygame module:

# importing libraries

import pygame

import time

import random

snake\_speed = 15

# Window size

window\_x = 720

window\_y = 480

# defining colors

black = pygame.Color(0, 0, 0)

white = pygame.Color(255, 255, 255)

red = pygame.Color(255, 0, 0)

green = pygame.Color(0, 255, 0)

blue = pygame.Color(0, 0, 255)

# Initialising pygame

pygame.init()

# Initialise game window

pygame.display.set\_caption('GeeksforGeeks Snakes')

game\_window = pygame.display.set\_mode((window\_x, window\_y))

# FPS (frames per second) controller

fps = pygame.time.Clock()

# defining snake default position

snake\_position = [100, 50]

# defining first 4 blocks of snake body

snake\_body = [[100, 50],

[90, 50],

[80, 50],

[70, 50]

]

# fruit position

fruit\_position = [random.randrange(1, (window\_x//10)) \* 10,

random.randrange(1, (window\_y//10)) \* 10]

fruit\_spawn = True

# setting default snake direction towards

# right

direction = 'RIGHT'

change\_to = direction

# initial score

score = 0

# displaying Score function

def show\_score(choice, color, font, size):

# creating font object score\_font

score\_font = pygame.font.SysFont(font, size)

# create the display surface object

# score\_surface

score\_surface = score\_font.render('Score : ' + str(score), True, color)

# create a rectangular object for the text

# surface object

score\_rect = score\_surface.get\_rect()

# displaying text

game\_window.blit(score\_surface, score\_rect)

# game over function

def game\_over():

# creating font object my\_font

my\_font = pygame.font.SysFont('times new roman', 50)

# creating a text surface on which text

# will be drawn

game\_over\_surface = my\_font.render(

'Your Score is : ' + str(score), True, red)

# create a rectangular object for the text

# surface object

game\_over\_rect = game\_over\_surface.get\_rect()

# setting position of the text

game\_over\_rect.midtop = (window\_x/2, window\_y/4)

# blit will draw the text on screen

game\_window.blit(game\_over\_surface, game\_over\_rect)

pygame.display.flip()

# after 2 seconds we will quit the program

time.sleep(2)

# deactivating pygame library

pygame.quit()

# quit the program

quit()

# Main Function

while True:

# handling key events

for event in pygame.event.get():

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_UP:

change\_to = 'UP'

if event.key == pygame.K\_DOWN:

change\_to = 'DOWN'

if event.key == pygame.K\_LEFT:

change\_to = 'LEFT'

if event.key == pygame.K\_RIGHT:

change\_to = 'RIGHT'

# If two keys pressed simultaneously

# we don't want snake to move into two

# directions simultaneously

if change\_to == 'UP' and direction != 'DOWN':

direction = 'UP'

if change\_to == 'DOWN' and direction != 'UP':

direction = 'DOWN'

if change\_to == 'LEFT' and direction != 'RIGHT':

direction = 'LEFT'

if change\_to == 'RIGHT' and direction != 'LEFT':

direction = 'RIGHT'

# Moving the snake

if direction == 'UP':

snake\_position[1] -= 10

if direction == 'DOWN':

snake\_position[1] += 10

if direction == 'LEFT':

snake\_position[0] -= 10

if direction == 'RIGHT':

snake\_position[0] += 10

# Snake body growing mechanism

# if fruits and snakes collide then scores

# will be incremented by 10

snake\_body.insert(0, list(snake\_position))

if snake\_position[0] == fruit\_position[0] and snake\_position[1] == fruit\_position[1]:

score += 10

fruit\_spawn = False

else:

snake\_body.pop()

if not fruit\_spawn:

fruit\_position = [random.randrange(1, (window\_x//10)) \* 10,

random.randrange(1, (window\_y//10)) \* 10]

fruit\_spawn = True

game\_window.fill(black)

for pos in snake\_body:

pygame.draw.rect(game\_window, green,

pygame.Rect(pos[0], pos[1], 10, 10))

pygame.draw.rect(game\_window, white, pygame.Rect(

fruit\_position[0], fruit\_position[1], 10, 10))

# Game Over conditions

if snake\_position[0] < 0 or snake\_position[0] > window\_x-10:

game\_over()

if snake\_position[1] < 0 or snake\_position[1] > window\_y-10:

game\_over()

# Touching the snake body

for block in snake\_body[1:]:

if snake\_position[0] == block[0] and snake\_position[1] == block[1]:

game\_over()

# displaying score continuously

show\_score(1, white, 'times new roman', 20)

# Refresh game screen

pygame.display.update()

# Frame Per Second /Refresh Rate

fps.tick(snake\_speed)

**Code Breakdown**

* **Initialization**: We initialize Pygame and set up the game window.
* **Colors and Dimensions**: Colors are defined for easy reference, and the game window size is specified.
* **Snake and Food Movement**: Movement is controlled with the arrow keys, and the snake's position is updated based on input.
* **Game Logic**: Includes collision detection, score update, and conditions for the game over scenario.
* **Main Game Loop**: Controls the overall flow of the game, checking for user input, updating the screen, and handling collisions.

This code should be run in an environment where you have Pygame installed. You can install Pygame using pip if you haven't already:

pip install pygame

With this guide and code, you should be able to create and play your own Snake game. Enjoy coding, and happy gaming!

**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 720x480 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.

**Output:**

