Snake Game in Python – Using Pygame module

Last Updated : 12 Aug, 2024

Snake game is one of the most popular arcade games of all time. In this game, the main objective of the player is to catch the maximum number of fruits without hitting the wall or itself. Creating a snake game can be taken as a challenge while learning Python or Pygame. It is one of the best beginner-friendly projects that every novice programmer should take as a challenge. Learning to build a video game is kinda interesting and fun learning.

We will be using **Pygame** to create this snake game. **Pygame** is an open-source library that is designed for making video games. It has inbuilt graphics and sound libraries. It is also beginner-friendly, and cross-platform.

Installation:

To install Pygame, you need to open up your terminal or command prompt and type the following command:

```
pip install pygame
```

After installing Pygame we are ready to create our cool snake game.

A step-by-step approach for creating a Snake Game using Pygame:

Step 1: First we are importing the necessary libraries.

- After that, we are defining the width and height of the window in which the game will be played.
- And define the color in RGB format that we are going to use in our game for displaying text.

Python

```python
# 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)
```

Step 2: After importing libraries we need to initialize Pygame using **pygame.init()** method.

- Create a game window using the width and height defined in the previous step.
- Here **pygame.time.Clock()** will be used further in the main logic of the game to change the speed of the snake.

Python

```python
# 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()
```

Step 3: Initialize snake position and its size.

- After initializing snake position, initialize the fruit position randomly anywhere in the defined height and width.
- By setting direction to RIGHT we ensure that, whenever a user runs the program/game, the snake must move right to the screen.

Python

```python
# 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
```

Step 4: Create a function to display the score of the player.

- In this function, firstly we're creating a font object i.e. the font color will go here.
- Then we are using render to create a background surface that we are going to change whenever our score updates.
- Create a rectangular object for the text surface object (where text will be refreshed)

- Then, we are displaying our score using **blit. blit** takes two argument **screen.blit(background,(x,y))**

Python

```
# 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)
```

Step 5: Now create a game over function that will represent the score after the snake is hit by a wall or itself.

- In the first line, we are creating a font object to display scores.
- Then we are creating text surfaces to render scores.
- After that, we are setting the position of the text in the middle of the playable area.
- Display the scores using **blit** and updating the score by updating the surface using flip().
- We are using sleep(2) to wait for 2 seconds before closing the window using quit().

Python

```python
# 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()
```

Step 6: Now we will be creating our main function that will do the following things:

- We will be validating the keys that will be responsible for the movement of the snake, then we will be creating a special condition that the snake should not be allowed to move in the opposite direction instantaneously.

- After that, if snake and fruit collide we will be incrementing the score by 10 and new fruit will be spanned.
- After that, we are checking that is the snake hit with a wall or not. If a snake hits a wall we will call game over function.
- If the snake hits itself, the game over function will be called.
- And in the end, we will be displaying the scores using the show_score function created earlier.

Python

```python
# 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)
```

Below is the Implementation

Python

```
# 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')
```

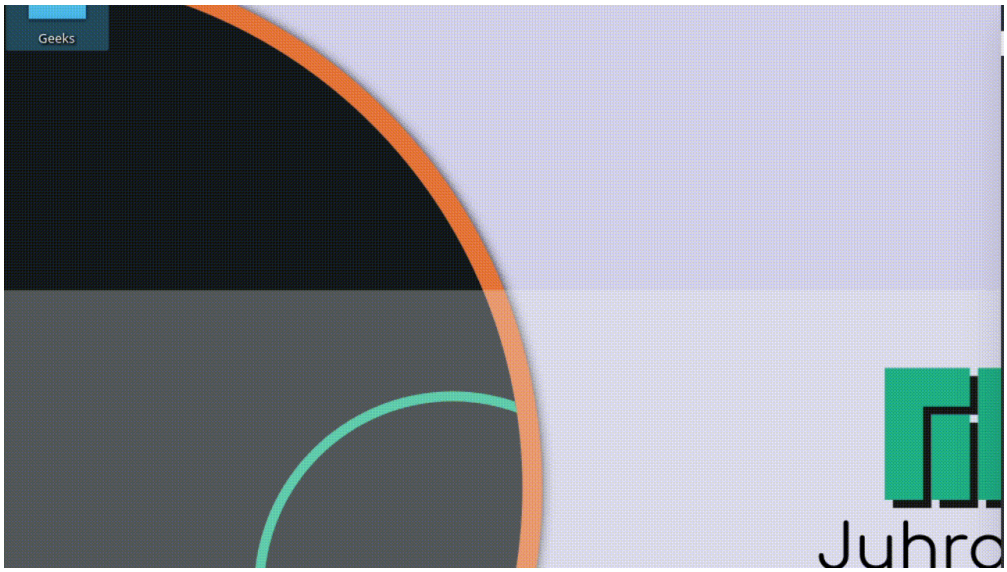
```
game_window = pygame.display.set_mode((window_x, window_y))

# 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)
```

Output:



Snake using Pygame

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.

Snake Game in Python – Using Pygame module – FAQs

How to Create a Snake Game Using Pygame

Creating a Snake game using Pygame involves setting up the game environment, handling user input, managing game logic, and updating the game state. Here's a high-level overview of the steps involved:

1. ***Initialize Pygame**: Set up the Pygame environment including the display window.*
2. ***Game Loop**: Implement a game loop that continuously updates the game state, renders the screen, and handles events.*
3. ***Handle Events**: Manage user input such as keyboard presses to control the snake's direction.*
4. ***Update Game State**: Move the snake, check for collisions, manage snake growth upon eating food, and handle game over scenarios.*
5. ***Draw/Graphics**: Update and render the snake, food, and possibly a score on the screen.*

To get started, you'll need to install Pygame, if you haven't already:

```
pip install pygame
```

Which Algorithm is Used for Snake Game in Python?

The Snake game typically doesn't require complex algorithms. The game mechanics are based on simple rules and operations like queue data structures where the head moves in a direction and the rest follows, segments get added to the tail, and boundary and self-collision checks. Advanced implementations might use pathfinding algorithms (like A or Dijkstra's algorithm) to create an AI that automatically plays the game.*

Is Snake Game Easy to Make?

Yes, making a basic Snake game is relatively easy and is often used as an introductory project for learning a programming language or a new library. The simplicity of the game mechanics makes it suitable for beginners, especially when using libraries like Pygame that simplify graphics and event handling.

What is Pygame Module in Python?

Pygame is a set of Python modules designed for writing video games. It provides functionalities like creating windows, drawing shapes, capturing mouse and keyboard actions, and playing sounds. With Pygame, programmers can create fully featured games and multimedia programs in Python.

How to Create Tic Tac Toe in Python

Creating a Tic Tac Toe game in Python can be done by managing a board as a 2D list, handling user input, and checking for win conditions. Here's a simple version of how you could implement it:

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_win(board, player):
    win_conditions = [
        [board[0][0], board[0][1], board[0][2]],
        [board[1][0], board[1][1], board[1][2]],
        [board[2][0], board[2][1], board[2][2]],
        [board[0][0], board[1][0], board[2][0]],
        [board[0][1], board[1][1], board[2][1]],
        [board[0][2], board[1][2], board[2][2]],
        [board[0][0], board[1][1], board[2][2]],
        [board[2][0], board[1][1], board[0][2]]
    ]
    return [player, player, player] in win_conditions

def tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]
    player = "X"
    while True:
        print_board(board)
        row = int(input(f"Enter the row for {player}: "))
        col = int(input(f"Enter the column for {player}: "))
        if board[row][col] != " ":
            print("Invalid move, try again.")
            continue
        board[row][col] = player
        if check_win(board, player):
            print_board(board)
            print(f"{player} wins!")
            break
```

```
            if all(all(row != " " for row in board_row) for board_row
in board):
                print_board(board)
                print("It's a draw!")
                break
            player = "O" if player == "X" else "X"

tic_tac_toe()
```

Looking to dive into the world of programming or sharpen your Python skills? Our **Master Python: Complete Beginner to Advanced Course** is your ultimate guide to becoming proficient in Python. This course covers everything you need to build a solid foundation from fundamental programming concepts to advanced techniques. With **hands-on projects**, real-world examples, and expert guidance, you'll gain the confidence to tackle complex **coding challenges**. Whether you're starting from scratch or aiming to enhance your skills, this course is the perfect fit. Enroll now and master Python, the language of the future!

Previous Article

Tic Tac Toe GUI In Python using PyGame

Next Article

8-bit game using pygame

Similar Reads

Adding Collisions Using pygame.Rect.colliderect in Pygame

Prerequisite: Drawing shapes in Pygame, Introduction to pygame In this article, we are going to use pygame.Rect.colliderect for adding collision in a shape usin...

3 min read

Snake Water Gun game using Python and C

Snake Water Gun is one of the famous two-player game played by many people. It is a hand game in which the player randomly chooses any of the three forms i....

8 min read

Snake Game Using Tkinter - Python

For many years, players all across the world have cherished the iconic video game Snake. The player controls a snake that crawls around the screen while...

12 min read

Create a Snake-Game using Turtle in Python

A snake game is an arcade maze game which has been developed by Gremlin Industries and published by Sega in October 1976. It is considered to be a skillfu...

8 min read

AI Driven Snake Game using Deep Q Learning

Content has been removed from this Article

1 min read

Brick Breaker Game In Python using Pygame

Brick Breaker is a 2D arcade video game developed in the 1990s. The game consists of a paddle/striker located at the bottom end of the screen, a ball, and...

14 min read

Dodger Game using Pygame in Python

A Dodger game is a type of game in which a player has access to control an object or any character to dodge (to avoid) upcoming obstacles and earn points...

5 min read

PyQt5 - Snake Game

In this article, we will see how we can design a simple snake game using PyQt5. Snake is the common name for a video game concept where the player...

14 min read

Stimulate bouncing game using Pygame

In this article, we will learn to make a bouncing game using Pygame. Pygame is a set of Python modules designed for writing video games. It adds functionality o...

4 min read

Angry Bird Game Using PyGame

Let's discuss the angry bird game using Pygame, I hope this article will be very interesting as it provides a holistic gaming experience at the end. We begin by...

14 min read

Article Tags : Python Python-projects Python-PyGame

Practice Tags : python