Python Project 02: Snake Game

1. Project Overview

In this project, we will build a complete **Snake Game** using Python and Tkinter.

The project has a clear separation between:

- **Frontend:** Game window, canvas, score, buttons and keyboard controls
- **Backend:** Snake movement, food generation, collision detection, score and game state

Tkinter's `Canvas` can be used to draw the game objects, while `bind()` handles keyboard events and `after()` schedules repeated game updates.

---

2. Features

- Snake movement
- Arrow key controls
- WASD controls
- Food generation
- Snake growth
- Score system
- Wall collision
- Self collision
- Game Over screen
- Restart button
- Modern dark UI
- Automatic game loop

---

3. Technologies

- Python 3
- Tkinter
- Object Oriented Programming
- Python `random` module

No external package is required.

---

4. Folder Structure

```
snake_game/
□
□□□ main.py
□□□ ui.py
□□□ game.py
□□□ README.md
```

---

5. How the Game Works

The game uses a grid.

The snake is stored as a list of coordinates:

```
[(10, 10), (9, 10), (8, 10)]
```

Every game update:

1. Calculate the next head position.

2. Check wall collision.
3. Check self collision.
4. Move the snake.
5. Check whether food was eaten.
6. Increase score and snake length if food was eaten.
7. Generate new food.
8. Redraw the board.

The UI displays the current state of the game.

---

6. Frontend Code

ui.py

```
import tkinter as tk
from game import SnakeGame


class SnakeUI:
    CELL_SIZE = 25
    BOARD_SIZE = 20

    def __init__(self, root):
        self.root = root
        self.root.title("Snake Game")
        self.root.resizable(False, False)
        self.root.configure(bg="#0f172a")

        self.game = SnakeGame(
            width=self.BOARD_SIZE,
            height=self.BOARD_SIZE
        )
```

```
        self.running = True
        self.speed = 120

        self.create_ui()
        self.bind_keys()
        self.draw()

        self.root.after(self.speed, self.game_loop)

    def create_ui(self):
        title = tk.Label(
            self.root,
            text="SNAKE",
            font=("Arial", 24, "bold"),
            fg="#38bdf8",
            bg="#0f172a"
        )
        title.pack(pady=(15, 5))

        self.score_label = tk.Label(
            self.root,
            text="Score: 0",
            font=("Arial", 14, "bold"),
            fg="white",
            bg="#0f172a"
        )
        self.score_label.pack(pady=(0, 10))

        self.canvas = tk.Canvas(
            self.root,
            width=self.BOARD_SIZE * self.CELL_SIZE,
            height=self.BOARD_SIZE * self.CELL_SIZE,
            bg="#020617",
```

```
            highlightthickness=2,
            highlightbackground="#334155"
        )
        self.canvas.pack(padx=20)

        self.message_label = tk.Label(
            self.root,
            text="Use Arrow Keys or WASD",
            font=("Arial", 11),
            fg="#94a3b8",
            bg="#0f172a"
        )
        self.message_label.pack(pady=10)

        self.restart_button = tk.Button(
            self.root,
            text="Restart Game",
            command=self.restart,
            font=("Arial", 11, "bold"),
            bg="#2563eb",
            fg="white",
            activebackground="#1d4ed8",
            activeforeground="white",
            relief="flat",
            padx=18,
            pady=8,
            cursor="hand2"
        )
        self.restart_button.pack(pady=(0, 18))

    def bind_keys(self):
        self.root.bind("<Up>", lambda event: self.change_direction("up"))
        self.root.bind("<Down>", lambda event: self.change_direction("down"))
        self.root.bind("<Left>", lambda event: self.change_direction("left"))
```

```
        self.root.bind("<Right>", lambda event: self.change_direction("right"))

        self.root.bind("w", lambda event: self.change_direction("up"))
        self.root.bind("s", lambda event: self.change_direction("down"))
        self.root.bind("a", lambda event: self.change_direction("left"))
        self.root.bind("d", lambda event: self.change_direction("right"))

    def change_direction(self, direction):
        if self.running:
            self.game.change_direction(direction)

    def game_loop(self):
        if not self.running:
            return

        if self.game.move():
            self.draw()
            self.score_label.config(
                text=f"Score: {self.game.score}"
            )
            self.root.after(self.speed, self.game_loop)
        else:
            self.running = False
            self.draw_game_over()

    def draw(self):
        self.canvas.delete("all")

        # Food
        food_x, food_y = self.game.food

        x1 = food_x * self.CELL_SIZE
        y1 = food_y * self.CELL_SIZE
        x2 = x1 + self.CELL_SIZE
```

```
        y2 = y1 + self.CELL_SIZE

        self.canvas.create_oval(
            x1 + 4,
            y1 + 4,
            x2 - 4,
            y2 - 4,
            fill="#f43f5e",
            outline=""
        )

        # Snake
        for index, (x, y) in enumerate(self.game.snake):
            x1 = x * self.CELL_SIZE
            y1 = y * self.CELL_SIZE
            x2 = x1 + self.CELL_SIZE
            y2 = y1 + self.CELL_SIZE

            self.canvas.create_rectangle(
                x1 + 2,
                y1 + 2,
                x2 - 2,
                y2 - 2,
                fill="#22c55e" if index == 0 else "#16a34a",
                outline=""
            )

    def draw_game_over(self):
        self.canvas.create_rectangle(
            80,
            190,
            420,
            310,
            fill="#0f172a",
```

```
            outline="#ef4444",
            width=2
        )

        self.canvas.create_text(
            250,
            225,
            text="GAME OVER",
            fill="#ef4444",
            font=("Arial", 26, "bold")
        )

        self.canvas.create_text(
            250,
            265,
            text=f"Final Score: {self.game.score}",
            fill="white",
            font=("Arial", 14)
        )

        self.message_label.config(
            text="Press Restart Game to play again"
        )

    def restart(self):
        self.game.reset()
        self.running = True
        self.message_label.config(
            text="Use Arrow Keys or WASD"
        )
        self.score_label.config(
            text="Score: 0"
        )
        self.draw()
```

```
        self.root.after(self.speed, self.game_loop)
```

---

7. Backend / Game Logic Code

game.py

```
import random


class SnakeGame:
    def __init__(self, width=20, height=20):
        self.width = width
        self.height = height
        self.reset()

    def reset(self):
        center_x = self.width // 2
        center_y = self.height // 2

        self.snake = [
            (center_x, center_y),
            (center_x - 1, center_y),
            (center_x - 2, center_y)
        ]

        self.direction = "right"
        self.next_direction = "right"
        self.score = 0
        self.game_over = False

        self.spawn_food()
```

```
  def change_direction(self, direction):
      opposite = {
          "up": "down",
          "down": "up",
          "left": "right",
          "right": "left"
      }

      if direction in opposite:
          if direction != opposite[self.direction]:
              self.next_direction = direction

  def move(self):
      if self.game_over:
          return False

      self.direction = self.next_direction

      head_x, head_y = self.snake[0]

      if self.direction == "up":
          new_head = (head_x, head_y - 1)

      elif self.direction == "down":
          new_head = (head_x, head_y + 1)

      elif self.direction == "left":
          new_head = (head_x - 1, head_y)

      else:
          new_head = (head_x + 1, head_y)

      if self.check_collision(new_head):
          self.game_over = True
```

```
            return False

        self.snake.insert(0, new_head)

        if new_head == self.food:
            self.score += 1
            self.spawn_food()
        else:
            self.snake.pop()

        return True

    def check_collision(self, position):
        x, y = position

        # Wall collision
        if x < 0 or x >= self.width:
            return True

        if y < 0 or y >= self.height:
            return True

        # Self collision
        if position in self.snake:
            return True

        return False

    def spawn_food(self):
        available_cells = [
            (x, y)
            for x in range(self.width)
            for y in range(self.height)
            if (x, y) not in self.snake
```

```
        ]

        if available_cells:
            self.food = random.choice(available_cells)
        else:
            self.food = None
            self.game_over = True
```

---

8. Main File

main.py

```
import tkinter as tk
from ui import SnakeUI


def main():
    root = tk.Tk()
    SnakeUI(root)
    root.mainloop()


if __name__ == "__main__":
    main()
```

---

9. How Frontend and Backend Connect

The connection is simple:

```
main.py
  
```

SnakeUI

SnakeGame

Game State

Snake Movement
Food
Collision
Score

SnakeUI redraws Canvas

ui.py does not decide whether the snake has collided.

That responsibility belongs to game.py.

The frontend only displays the result.

---

10. How to Run

Open the project folder in terminal:

```
cd snake_game
```

Run:

```
python main.py
```

No external package installation is required.

---

11. Basic Testing

Test these cases:

- Press arrow keys and confirm the snake moves.
- Press `W`, `A`, `S`, `D` and confirm movement.
- Eat food and check that the score increases.
- Confirm that the snake grows after eating food.
- Hit the wall and confirm Game Over.
- Make the snake hit itself and confirm Game Over.
- Try reversing direction immediately.
- Press Restart Game and confirm that the score and snake reset.

---

12. Small Improvements

After completing the basic version, students can add:

- Increasing speed after every few points
- High score system
- Pause button
- Sound effects
- Different difficulty levels
- Start screen
- Game history
- Different snake themes
- Obstacles
- Multiple food types

Project Goal

The main purpose of this project is not only to create a Snake Game.

It is to practice how a real application can be divided into:

UI Logic State Events Output

This same separation becomes useful when building larger applications.

haas.dev

https://dev-roast-app.vercel.app

Visit haas.dev for more resources and guides.