🐍

**Snake 'n' Shake**

**Introduction**

This document provides an overview of the design and implementation of the Snake Game using Python's Turtle graphics library. The game includes a snake controlled by the player, food items to eat, a scoring system, lives, and a start screen. The project demonstrates the utility of the dynamic array (Python list internally) data structure .

**WHAT this Documentation contains?**

1. Overview of the Turtle Library in Python.
2. Data structure used to build the GAME.
3. Program Design.
4. Game flow
5. Source code
6. Class implementation

**PYTHON TURTLE LIBRARY**

The Python Turtle library is a standard graphics library that allows users to create drawings and simple animations in a fun and interactive way.

### **Advantages of Using the Turtle Library**

1. **Beginner-Friendly**: The Turtle library is perfect for beginners as it provides a gentle introduction to programming concepts through visual feedback.
2. **Visual Learning:** It helps users learn through visualization, which is particularly useful in understanding loops, angles, and coordinates.
3. **Immediate Feedback**: Changes in the code result in immediate visual updates, helping users understand the effect of their code quickly.
4. **Integration with Python**: Since it's part of Python’s standard library, no additional installations are required, making it easy to start using immediately.
5. **Supports event-driven programming**: Turtle allows users to handle events like key presses and mouse clicks, introducing interactive elements to projects.
6. **Cross-Platform**: The Turtle library works across different operating systems, including Windows, macOS, and Linux, ensuring accessibility for all users.

Data structures used!

**DATA STRUCTURE USED**

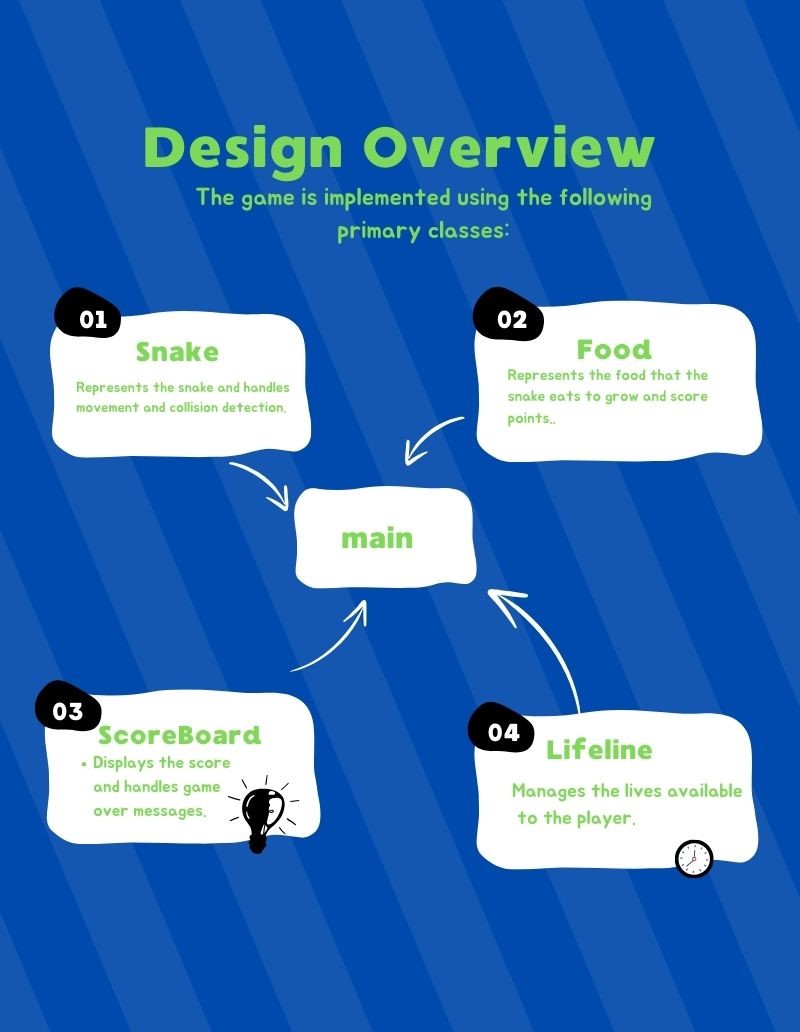
The concept of Dynamic arrays is used for creating snake bodies, and internally a Python list is being used.

**Why: DYNAMIC ARRAYS?**

Dynamic arrays are a linear data structure that can grow to any size as per the user's desire by resizing themselves once

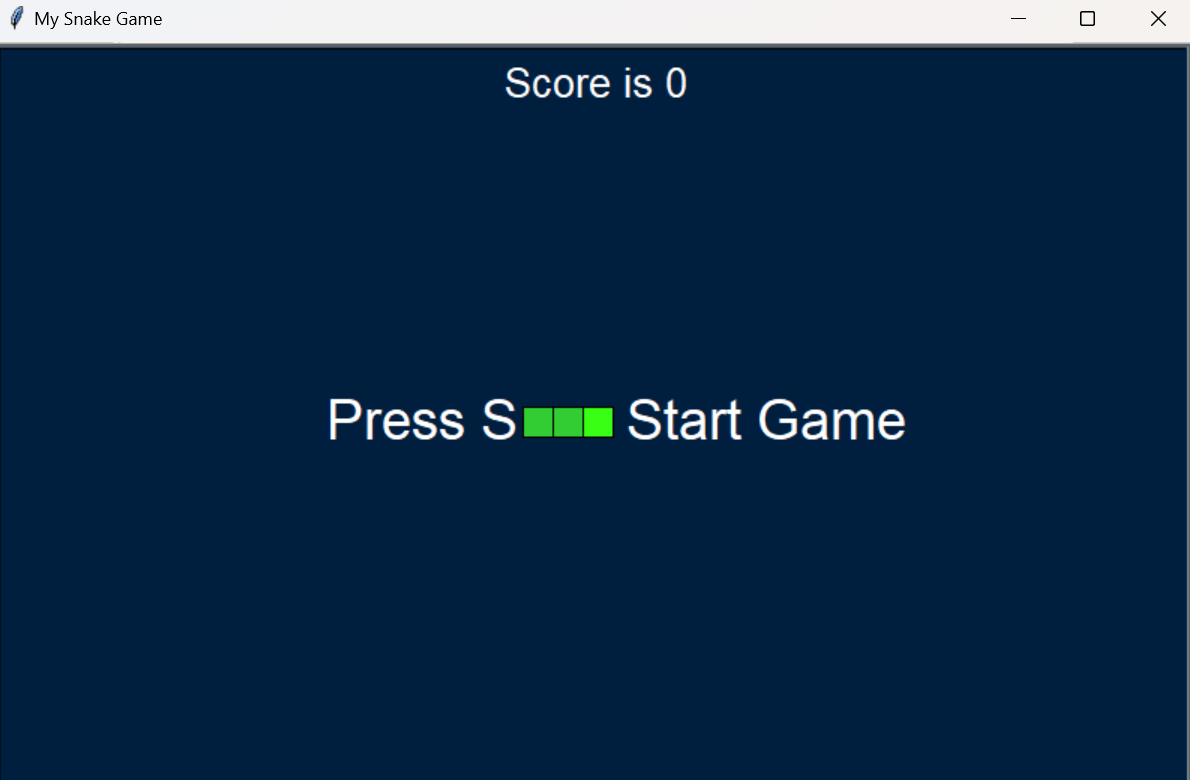
the capacity is met.

Since the snake’s body grows as it eats fruits and its entire body moves together, dynamic arrays are best for its implementation!



### **Game Flow**

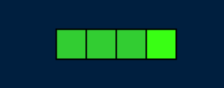
1. The game begins with a landing page, displaying the "Press S to Start" message.



1. Upon pressing "S", the game starts and the snake becomes controllable using the arrow keys.



1. The snake moves automatically; it grows when it eats food and scores points.

 +  → 

1. The game checks for collisions with walls, itself, and food.

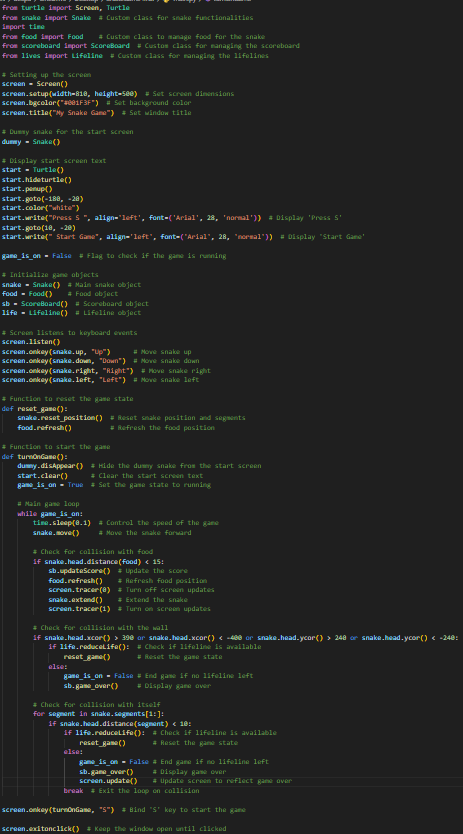


1. If a collision occurs, a life is lost. If all lives are lost, the game ends.

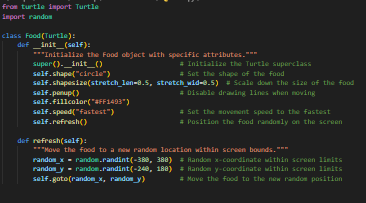


**Source code**

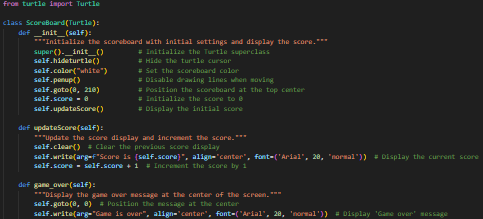
* **main.py**

****

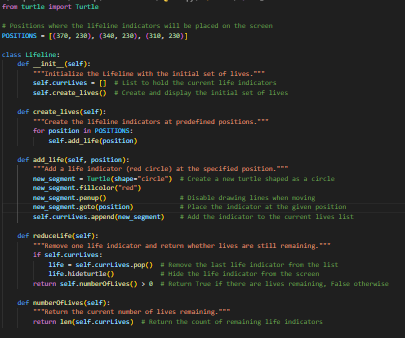
* **food.py**

****

* **Scoreboard.py**

****

* **lives.py**

****

**Class Implementations**

## Snake Class

* **Attributes**:
  + segments: A list of Turtle objects representing the snake's body.
  + head: The first segment of the snake.
* Methods:
  + \_\_init\_\_(): Initializes the snake with three segments.
  + create\_snake(): Creates the initial snake.
  + add\_segment(position): Adds a new segment to the snake.
  + move(): Moves the snake forward.
  + extend(): extends the snake when it eats food.
  + reset\_position(): Resets the snake's position and direction after losing a life.
  + disAppear(): Hides the snake (used for the start screen).
  + up(), down(), left(), right(): Controls the direction of the snake.

### Food Class

* Attributes:
  + Inherits from Turtle.
* Methods:
  + \_\_init\_\_(): Initializes the food with a random position.
  + refresh(): Place the food at a new random position.

### 

### ScoreBoard Class

* Attributes:
  + Inherits from Turtle.
  + score: Keeps track of the current score.
* Methods:
  + \_\_init\_\_(): Initializes the scoreboard and displays the initial score.
  + updateScore(): Updates and displays the score.
  + game\_over(): Displays the game-over message.

### Lifeline Class

* Attributes:
  + currLives: A list of Turtle objects representing lives.
* Methods:
  + \_\_init\_\_(): Initializes the lifeline with three lives.
  + create\_lives(): Creates the initial lives.
  + add\_life(position): Adds a life indicator to the screen.
  + reduceLife(): Removes a life after a collision and checks if lives remain.
  + numberOfLives(): Returns the number of lives.

## 

## 

## 

## 

## 

# Conclusion

The snake game leverages fundamental data structures such as lists and tuples to efficiently manage game state and behavior. The dynamic nature of lists allows for the flexible growth of the snake and easy tracking of player lives, while tuples provide a reliable method for storing and comparing positions.