Group: Lediya Solomon & Emmanuel Ephraim
INST326
12/16/23

<u>What is our project?</u>

The project consists of a playable Snake game where users control a snake icon to eat food items and gain length/points while avoiding collisions. Object-oriented Python code defines Snake, Food, and Scoreboard classes to model different elements of arcade-style gameplay. Unit testing with mocking and assertions acts as automatic verification of expected behavior as modifications are made.

Running and Using the Application

- Execution Entry Point: main.py launches the full Pygame application
- Input Method: Arrow keys control snake direction
- Output: Graphical window with snake, food, score, walls rendered
- Persistence: High score stored/retrieved from local data.txt file
- Goal: Maneuver snake to eat randomly positioned food to gain score
- Lose Condition: Snake collides with wall or own body

Unit Testing Features

- Test Framework: Python's unittest
- Test Execution: Run test_snake.py
- Test Double: Mocking and patch simulate scenarios
- Test Cases:

Validate snake movement and length growth
Check food repositioning works
Confirm score incrementing
Verify file I/O for high score

- Test Adequacy: 80%+ code coverage targeted
- Ensure critical game behaviors function properly

Implementation Plan

- Phase 1: Core game mechanics and Pygame rendering
- Phase 2: Class design and game object behaviors
- Phase 3: Unit test coverage of key methods

- Phase 4: Refinements and polish based on testing feedback

So in summary, main.py launches the full application, test_snake.py runs all defined unit tests, and you can drill down to specific test cases as needed. Automated testing allows validation of expected functionality.

To enable persistence of the top score achieved across multiple gaming sessions, the high score value will be read from and written to a basic text file. When the Scoreboard module is initialized, it will attempt to open a local file called "data.txt" and read the contents, which should be a numeric string representing the historical high score. This value gets converted to an integer and stored as an attribute within the Scoreboard. Later in runtime, if a higher score is reached, the Scoreboard writes the new high score back to "data.txt" by overwriting the file contents. This allows the maximum score to be loaded and preserved between separate instances of running the Snake game application, enabling a persistent high score that the user can strive to improve over time. Text file-based persistence offers a lightweight way to retain the top score data without needing a database or server API backend.

Documentation on how to use the program / how to interpret the output of the program:
The game output is simply the visual Snake game screen, showing the snake, food, walls, and current score. Use the up, down, left, and right arrow keys to control the direction of the snake. The goal is to maneuver the continuously moving snake to collide with (eat) the food items that randomly spawn to make the snake longer and accumulate points in the score counter. Run ends when the user exits program.

Bibliography

"Python Arcade Library and Examples", Python Arcade Library,
https://api.arcade.academy/en/latest/index.html.

Used for game event handling, drawing shapes and textures, and keyboard input. Enabled core gameplay functionality.
"Pygame Snake Tutorial", Tech With Tim,
https://www.techwithtim.net/tutorials/game-development-with-python/snake-pygame/tutorial/.

Referred to for structure of game loop, detecting keypresses, and collision mechanics.
"Adding a Scoreboard to Snake Game in Pygame", Game Development Center,
https://gamedevcenter.com/adding-scoreboard-to-snake-game-in-pygame/.

We used as a guide for implementing the scoreboard UI and tying it to the game events.