The third solution is upgrading my Snake Game by integrating it with the Go programming language. Here's the pseudocode of how I plan to implement it:

1. **Game Initialization:**
   * Initialize core game variables such as score, frame rate, and segment size.
   * Define game entities using structs.
   * Create methods associated with these structs to handle their behaviour.
   * Set up the rendering system, potentially using a third-party package for handling graphics.
2. **Main Game Loop:**
   * Implement a loop that runs while the game is active:

a. **User Input Handling:**

* + - * Capture and process keyboard events to control the snake.
      * Include functionality to handle game exit commands.

b. **Update Game State:**

* + - * Call the snake's move method to update its position.
      * Check for collisions with itself, the game borders, or food.
      * On food collision, increase the score, call the grow method, and generate new food.

c. **Draw Game:**

* + - * Clear the screen.
      * Draw the game entities (snake and food).
      * Update the display with the new game state.

d. **Adjust Frame Rate:**

* + - * Modify the frame rate based on the game progression to increase difficulty.

1. **Game Over:**
   * Display the final score.
   * Clean up resources, such as closing the graphics context.
   * Exit the game.

Pseudocode for some critical functions using this programming language include:  
  
**1. CreateSnake**

* + Initialize the snake with a starting size and position.
  + Set the initial direction of the snake (e.g., moving right).

1. **ChangeDirection**
   * Take user input to change the snake's direction.
   * Ensure the new direction is not directly opposite to the current direction (to prevent the snake from turning back on itself).

**3. MainGameLoop**

* + Run an infinite loop.
    - Handle user input (e.g., change direction).
    - Move the snake.
    - Check for collisions (with the wall, self, or food).
    - Draw the updated game state.
    - Adjust the loop speed based on the frame rate.

The base code I uploaded from RosettaCode ran with no issues and printed a “Final Score:” after I removed irrelevant code, initialized the go module and ran “go build”.

A screenshot of a computer program

Description automatically generated

Then realized that the scoring system is not as accurate as I thought. When the game is quit (using ESC key) and the snake hits the boundaries, the final scores are displayed as “1” and “2” respectively.

A black background with white text

Description automatically generated

I fixed this error by creating a score field in Snake struct and initialized it to 0.

A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A logo of a company

Description automatically generated

Like my other programming languages, I have implemented a x2 speed increase after every 15 seconds to increase game difficulty.

A computer screen shot of a program code

Description automatically generated

There is also an input for Student ID with validation before the game begins, printing an error message if the Student ID is not 8 digits.

A computer screen shot of code

Description automatically generated

TESTING

Here are some tests I ran to ensure the game was working to my expectations:

* 1. Basic Gameplay

A blue and white rectangle

Description automatically generated

* 1. Snake Length Increase

A screen shot of a game

Description automatically generated

* 1. Final Score

A close up of numbers

Description automatically generated

* 1. Student ID Input with Validation

A screen shot of a computer

Description automatically generated