The second solution I’m enhancing my Snake Game with is using C programming language and the pseudocode is as follows:

1. **Game Initialization:**
   * Initialize game variables: score, frame rate, and segment size.
   * Initialize game entities: Snake and Food.
     + Use structures (**struct**) for Snake and Food entities.
   * Set up a rendering system.
2. **Main Game Loop:**
   * While the Snake is alive:

a. **User Input Handling:**

* + - Capture and process keyboard input for arrow keys to change the snake's direction.
    - Handle quit events (like pressing the ESC key or closing the window).

b. **Update Game State:**

* + - Update the snake's position based on its current direction.
    - Handle snake body movement and check for self-collisions.
    - Detect collisions with food: increase score, grow the snake, and spawn new food.

c. **Draw Game:**

* + - Clear the screen (buffer).
    - Draw the snake and food on the screen.
    - Refresh the display to show the updated game state.

d. **Adjust Frame Rate:**

* + - Increase the frame rate at certain intervals or based on score to make the game progressively more challenging.

1. **Game Over:**
   * Display the final score.
   * Free any dynamically allocated resources.
   * Exit the game and close any graphics/window systems in use.

The pseudocode for some critical functions is also listed below:

**Update Snake Movement**

1. Calculate the new head position based on the current direction.
2. Check if the snake is growing:
   * If yes, add a new segment at the new head position without moving the last segment.
   * If not growing, move each segment to the position of the segment in front of it.
3. Set the first segment (head) to the new position.

**Collision Check**

1. **Check Wall Collision:**
   * If the snake's head position is outside the **gameArea**, return a Wall Collision status.
2. **Check Self Collision:**
   * Iterate through the snake's body segments:
     + If the head's position matches any of the body segments, return a Self-Collision status.
3. **Check Food Collision:**
   * If the snake's head position matches the food position:
     + Return a Food Collision status.

The base code has been implemented from RosettaCode which utilizes the ‘ncurses’ library for handling terminal based display and input.

A screen shot of a game

Description automatically generated

I removed the unnecessary code (e.g. running on Linux) and realized the navigation had incorrect mapping when it ran in the terminal ( J = up, L = down, K = left, I = right) even though it stated different in the code.

A screen shot of a computer code

Description automatically generated

There also seemed to be an issue with the code not updating each time it was saved so I created a “./run\_snake.sh” build file to execute so “./run\_snake.sh” is used to run the game in the terminal instead of just “./snake”.

A screenshot of a computer program

Description automatically generated

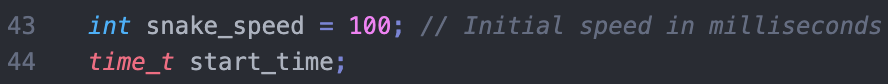
*#include* <ncurses.h>

A screen shot of a computer program

Description automatically generated

I further corrected the base code by adding the #include <ncurses.h> header and changing the input to accept arrow keys instead. This was also put into “void handle\_input” for better handling and then into the main class.

I have been trying to implement border wrapping like I did in Python but was unsuccessful, making the game harder anyways. However, I did modify the game to get faster by x2 speed every 15 seconds.



A screen shot of a computer code

Description automatically generated

In terms of uniqueness, the snake game asks for the users Student ID before the game starts with validation of having only 8 digits.

A computer screen shot of text

Description automatically generated

TESTING:  
These are the tests I ran to ensure the C program was working as expected:

* 1. Basic Gameplay

A black background with white dots

Description automatically generated A screenshot of a computer

Description automatically generated

* 1. Snake Length Increase

A black and white screen shot of a letter j

Description automatically generated

* 1. Game Over

A screenshot of a computer

Description automatically generated

* 1. Student ID Validation

A black background with white text

Description automatically generated

