COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Toluwatomi Emoruwa & Victor Trepanier

Team Members Evaluated Aashvi Patel & Jhanvi Shah

Part I: OOD Quality

[6 marks] *OOD is about sensible code modularization. Looking at the header files of each object, can you easily interpret the possible behaviours of the objects involved in the program, and how they would interact with each other in the program? Comment on what you have observed, both positive and negative features.*

Positive Features:

- The header files provide clear class declarations for objects like GameMechs, Player, objPos, and objPosArrayList.
- Each object seems to encapsulate specific behaviors and functionalities related to the game.

Negative Features:

- There is room for cleaner documentation, the removal od unnecessary commented lines in the header files and more detailed comments to explain complex interactions or relationships between objects and increase readability of the file in total.

[6 marks] *Examine the main logic in the main program loop. Can you easily interpret how the objects interact with each other in the program logic through the code? Comment on what you have observed, both positive and negative features.*

Positive Features:

- The main loop clearly interacts with GameMechs and Player objects.
- Objects interact in a logical sequence: getting input, running logic, updating the game state, and drawing the screen.

Negative Features:

- Some features could have been better implemented by encapsulating them into features of the objects such as the Food Collision Check and Score Incrementation.
- Furthermore, all screen altering functions/code should be put in the Draw Screen function, for organisation's sake; this design principal wasn't adhered.

- [Suggestion for Streamlined Code]: Lines of Code could also have been reduced as there is redundant initialisation. An example is the creation of the GameMechPtr as a global variable but value allocation inside the initialise function.
- Blocks commented unused code need to be deleted.

[5 marks] *Quickly summarize in point form the pros and cons of the C++ OOD approach in the project versus the C procedural design approach in PPA3.*

OOD Approach Comparison (C++ vs. Procedural C):

C++ OOD Pros:

- Encapsulation of functionalities within classes improves modularity.
- Easier to manage behaviors and interactions between snake body, food and game boarders.

C++ OOD Cons:

- Potential complexity in object interactions and relationships might need clearer documentation. Such as figuring out the syntax for internal interactions between the player acting as an array list and food.
- Furthermore, understanding OOP concepts might pose a steeper learning curve for programmers new to object-oriented concepts.

Procedural C Pros:

- Simplicity in design might be easier to understand for straightforward applications.
- Singular Document format, although long, allowed readers to understand and locate parts of the code at a glance in the VScode GUI.

Procedural C Cons:

- Difficulty in managing larger or more complex systems due to lack of encapsulation and modularity.
- Lack of encapsulation and modularity can lead to difficulty in managing and maintaining larger codebases, making it less scalable as the project grows (such as trying to implement the Above and Beyond Features of the Food List System).
- Plus, Blocks of Object Definitions and Initialisations reduce file readability without proper formatting.

In summary, this project's adoption of Object-Oriented Design offers advantages in terms of encapsulation, modularity, code reusability, and code organization. However, it introduces complexities in managing object relationships and might require a learning curve for those unfamiliar with OOP.

Contrastingly, the procedural design in the previous assignment (PPA3) in C allowed for simplicity, ease of understanding, and efficiency in smaller-scale projects. Yet, it lacked the modularity, reusability, and scalability that OOP provides, making it less suitable for larger and more complex systems.

Part II: Code Quality

[5 marks] *Does the code offer sufficient comments, or deploys sufficient self-documenting coding style, to help you understand the code functionality more efficiently? If any shortcoming is observed, discuss how you would improve it.*

The code includes comments but are a lot of areas that lack sufficient explanations, especially in complex logic or interactions between objects such as in the Run Logic Section. The code overall contains a lot of unnecessary comments as mentioned before and little to no comments inside functions detailing their methodology of operation leaving the reader to infer functionality.

Improvement can be made by adding more descriptive comments or using self-documenting code and deleting unnecessary commented code. Furthermore, Portions of code can be sectioned off and streamlined with comments to improve organisation and readability.

[4 marks] *Does the code follow good indentation, add sensible white spaces, and deploys newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.*

The code mostly follows good indentation and spacing practices, helping readability. There is sufficient spaces demarcating sections of code.

There are noticeable inconsistencies in spacing or indentation (a lack of spacing specifically in some functions) within certain files, which could be improved/reformatted to for better consistency and clarity.

Part III: Quick Functional Evaluation

[8 marks] *Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just a technical user feedback)*

1. Terminal Remains Empty Until Input:

Possible Cause: This behavior suggests a lack of initial screen drawing or a delay in the display before taking any input.

Debugging Approach:

- Check the initialization sequence to ensure the game screen is properly drawn at the start.
- Verify if there's any delay or pause in the display before accepting user input.

2. Snake Body Movement and Appearance:

a. Incremental Movement:

Possible Cause: The snake might move only when individual keys are pressed instead of continuous movement.

Debugging Approach:

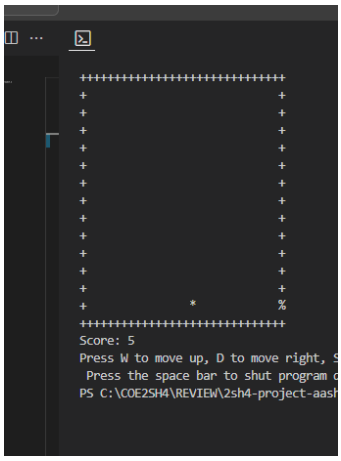
- Review the logic responsible for snake movement to ensure it allows continuous and smooth movement in the intended direction.
- Check if the game loop is correctly updating the snake's position based on the chosen direction.

b. Improper Wraparound and Display Inside Border:

Possible Cause: Incorrect boundary conditions or logic causing the snake or characters to display improperly.

Debugging Approach:

- Inspect the boundary checks and update conditions to ensure proper wraparound and character display within the borders.
- Verify that the snake's position coordinates are correctly handled when reaching the edges of the game board.



3. Snake Body Growth and Food Generation:

a. Snake Body Growth:

Possible Cause: Iteration 3 was never reached – snake object was never replaced with object array.

b. Improper Food Generation:

Possible Cause: Generation of food outside the game board's boundaries.

Debugging Approach:

- Review the logic for detecting food consumption and subsequent growth of the snake's body to ensure it functions correctly. As character updating tends to be sluggish so order of execution might be incorrect.
- Adjust the food generation limits within the generation function to ensure food is spawned within the valid board positions.
- Print the coordinates of the generated food to track its position and verify its correct functionality.

General Debugging Approach: Use print statements or logging to trace the different inputs and locations of gameboard items.

[6 marks] *Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause of the memory leakage.*

From the proofing the documents I see no evidence of memory leakage, all variables/objects initialised with “new” on the heap where properly deleted in the Cleanup Function. The Snake Games causes no memory leaks.

Part IV: Your Own Collaboration Experience (Ungraded)

Tell us about your experience in your first collaborated software development through this project – what was working and what wasn't. If you are a one-person team, tell us what you think may work better if you had a second collaborator working with you.

Our experience working on the together was intuitive, having a partner to consult and bounce idea off was crucial in code development and especially in debugging.