COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Meshak Sharma (marcy-reunion)

Team Members Evaluated ricas05 whitee24 (control-alt-delete)

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **25 marks**. Do not exceed 2 paragraphs per question.

Peer Code Review: OOD Quality

1. **[3 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.

The main loop incorporates two distinct objects, GameMechs and Player. The GameMechs object handles all of the game's mechanics such as the input and score. The Player object handles all features linked with the snake such as movement and direction.

Looking deeper into the code, they also implement two further classes: objPos and objPosArrayList. The objPos object helps keep track of any game positions for the snake and food. The objPosArrayList helps implement the movement of the snake. These classes as well as GameMechs are encapsulated within the Player class, to help with movement, and it's interaction with the game's features. This being said, this encapsulation can only be done when looking deeper into the various files of the code. From the main loop, we cannot see how the objects interact with each other at all.

Overall though, the code is quite easy to interpret and understand. Each function in the main code is concise and efficient, utilizing the various objects to carry out the background operations so that it is clear what each section of the main loop is doing.

2. **[3 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.

Pros with the C++ OOD Approach

- We were able to section off each main chunk of logic or features into different subsystems (objects), so it was simpler to implement new features or functions into a certain aspect of our program by just creating a new function into its designated class. This also helps with the clarity of the overall program.
- Object Oriented Design also helps us manage multiple pieces of information at a time related to a specific feature. It also helps us include functions that can work with this information without risk of altering the values.

Cons with the C++ OOD Approach:

- Working with multiple objects is much more complex, and requires a lot more setup than the more procedural approach we followed working in C.

- Furthermore, because OOD protects its values, working with and accessing these values in other parts of the code became much more challenging and complex.

Peer Code Review: Code Quality

- **[3 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 offers a concise and simple comment before each chunk of logic in the code, which clearly defines what each portion of the code does. The comments are not overdone to clutter up the code, yet appears enough so that the code is properly explained.

In functions where no comments are given (for example the RunLogic function), the code is short and easy to understand due to the clarity of the variable names used.

The only shortcoming found were the lack of comments in the objPosArrayList cpp file. While some of these functions are self-explanatory, some comments for clarification would be appreciated.

1. **[3 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.

Throughout the code, good and consistent indentation was used. The code isn't over indented and utilised when necessary. There is sufficient whitespace between main pieces of logic in the code, and for readability within functions. They also deploy newlines sensibly between different portions of a function, which enhances readability.

Peer Code Review: Quick Functional Evaluation

1. **[3 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 technical user feedback)
   The game offers a smooth bug free playing experience. The player/snake moves smooth and responds to the input from user. One thing I noticed is that the player moves a bit slower when moving from left to right compared to up and down. A possible cause to this could be the fact that the board is longer vertically and shorter horizontally. These properties could impact the movement of the snake and therefore influence the speed of the snake. A potential debugging approach could be to use the debugging tool within VS code and track the position of the snake's head and how fast it is being updated. Compare the speed of the snake when its moving vertical vs horizontal, this approach could lead you to why this scenario may be happening.

2. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.
   There was no memory leaks detected.

<u>Project Reflection</u>

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?
   Yes, I think it is sensible. The additional Pos struct allows it to be used in different areas of the class seamlessly. It also provides a cleaner method of coding, as there are not many duplicate parts of the same code. This maintains the simplicity of the script and allows for easier debugging.

2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. <u>You are expected to facilitate the discussion with UML diagram(s).</u>
   Another approach you could take in this project that would be relatively counterintuitive is removing the Pos struct. The reason this is counterintuitive is because although you are making the code a little less complex, you are sacrificing the modularity of the code. The Pos struct can be seen as a central hub for the position values, and by removing it you would have to now duplicate the logic for it in every instance you need it. Also, by having the pos struct you can add any additional attributes to it you can simply update it. By removing the struct you would have to manually adjust the objPos class itself.

objPos
+ x: int + y: int + symbol: char
+ objPos () + objPos (xPos:int, yPos: int, sym:char) + setObjPos (o:objPos): void + setObjPos (xPos:int, yPos:int, sym: char) : void + getObjPos () : objPos const + getSymbol () : char const + isPosEqual (refPos:const objPos*) : bool const + getSymbolIfPosEqual (refPos:const objPos*): char const