COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members _____khatiy2_____ ___1 member team_____

Team Members Evaluated _____verdinrd __shahinia __ (two-semantic-errors)

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 **30 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.
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.

Peer Code Review: Code Quality

1. **[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.
2. **[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.

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)
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.

Project Reflection

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

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?
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. You are expected to facilitate the discussion with UML diagram.

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.

When examining the code, it's generally straightforward to understand how the objects are interacting within the program's logic. The use of local variables such as height/width/player position to reduce function calls and improve readability go a long way into making Draw screen simple to understand. However, one area for improvement is the lack of inline comments, which would help clarify the intentions behind key parts of the code. This would be particularly helpful in understanding the global board variable, whose purpose is not immediately apparent. From context, it appears to represent the state of each position on the board, but this is not explicitly documented in the code.

Overall, the code is well-structured and readable but adding comments and improving the clarity around certain global objects would make it even easier to interpret and maintain.

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:

- Having modular code with behaviors/data members helps a lot in being able to maintain and manage complex systems/algorithms as it helps isolate bugs to only specific parts of code that can be changed easily and don't require extensive changes elsewhere.
- Code can easily be repurposed (ex: objPos class) to save time.
- Being able to change parameters of the game easily (like boardsize, player symbol, etc)
- Code is very scalable, adding new features (different food types, different player behaviors) is much simpler due to polymorphism and inheritance.
- Code Clarity: Cleaner organization with classes like Snake, Board, and Game.

Cons:

- Increased Complexity, need to carefully plan how objects work and interact with each other.
- Need good understanding of DMA (PPA3 did not use DMA).
- PPA3 was much easier to prototype, as procedural code typically takes less time to write

3. **[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.

   No, the code has no comments outside of the given template comments. Due to OOD design, the code is still easy enough to read, but I would definitely add more comments in the main program loop (especially within the draw screen function) as it could be quite confusing to understand what is going on unless you are already familiar with the program.

4. **[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.

   The code is well-indented, with blocks properly nested within their respective functions and constructs. nested loops are clearly separated visually. The code includes some whitespace to separate logic blocks, making it easy to follow. Functions and major logical segments are also separated by newlines, aiding readability. Overall, the code is well formatted in terms of readability.

3. **[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)

   Yes, upon 20 or so minutes of playing alongside checking the code, it appears that no bugs can be identified and the game runs smoothly.

4. **[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.

   All allocated items on the heap are deallocated at the end of the program, destructors are correctly implemented, no memory leak can be observed on dr. memory.

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

5. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

   Introducing the Pos struct adds an extra layer of complexity, making the design more bloated than necessary for a simple coordinate system. For example, you now need to dynamically allocate and deallocate Pos objects, which increases the risk of memory leaks or segmentation faults. A simpler design could have stored x and y directly within objPos. Many objPos methods repeatedly access pos->x and pos->y, which could have been avoided if x and y were directly part of objPos. The design of objPos requires implementing a custom copy constructor and assignment operator due to the dynamic allocation of Pos. This introduces extra development effort and increases the likelihood of bugs if these special member functions are improperly implemented.

6. **[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.</u>

   The objPos class can be simplified by directly storing x, y, and symbol as member variables instead of using a dynamically allocated Pos struct. This approach removes the need for dynamic memory management, reducing overhead and simplifying the implementation. With no dynamically allocated memory, the default copy constructor and assignment operator will work correctly without requiring custom implementations. Additionally, direct access to x and y makes the code more straightforward and easier to understand.