COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members **Tae Yeon Ha, Shaun Plassery**

Team Members Evaluated **Tae Yeon Ha, Shaun Plassery**

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.
   Yes, I can. They are all in the single class in the file and ordered in a good chronological sense. Functions are prototyped well in the header files for easy readability.
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.
   - **Encapsulation: Combines data (pos, symbol) and behavior into classes, promoting modularity.**
   - **Dynamic Memory Management: Supports dynamic allocation with proper cleanup using constructors/destructors.**
   - **For a developer its easier to work when everything is modularized and if you mess up it won't affect the rest of the code.**
   - **Robustness: Includes exception handling (e.g., std::out_of_range) for safer operations.**

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.
   - **Comments are done pretty well. In some files there are a bit less comments than ideal however I don't think it is hard to understand nevertheless. In the future, It would be gerat to add more general block comments on what each function does, its output and its significance in the code**
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.
   - **Yes, the code contains consistent indentation which improves readability, and spacing between lines to ensure it is not one jumbled mess. One potential drawback is that some of the conditional statements in the program were very long and fit into one line. This can be fixed by separating the statement into multiple lines for cleaner code.**

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)

- **It is relatively non buggy. However due to it running in the terminal and it refreshing the screen 100,000ms there will be some lag in the game.**

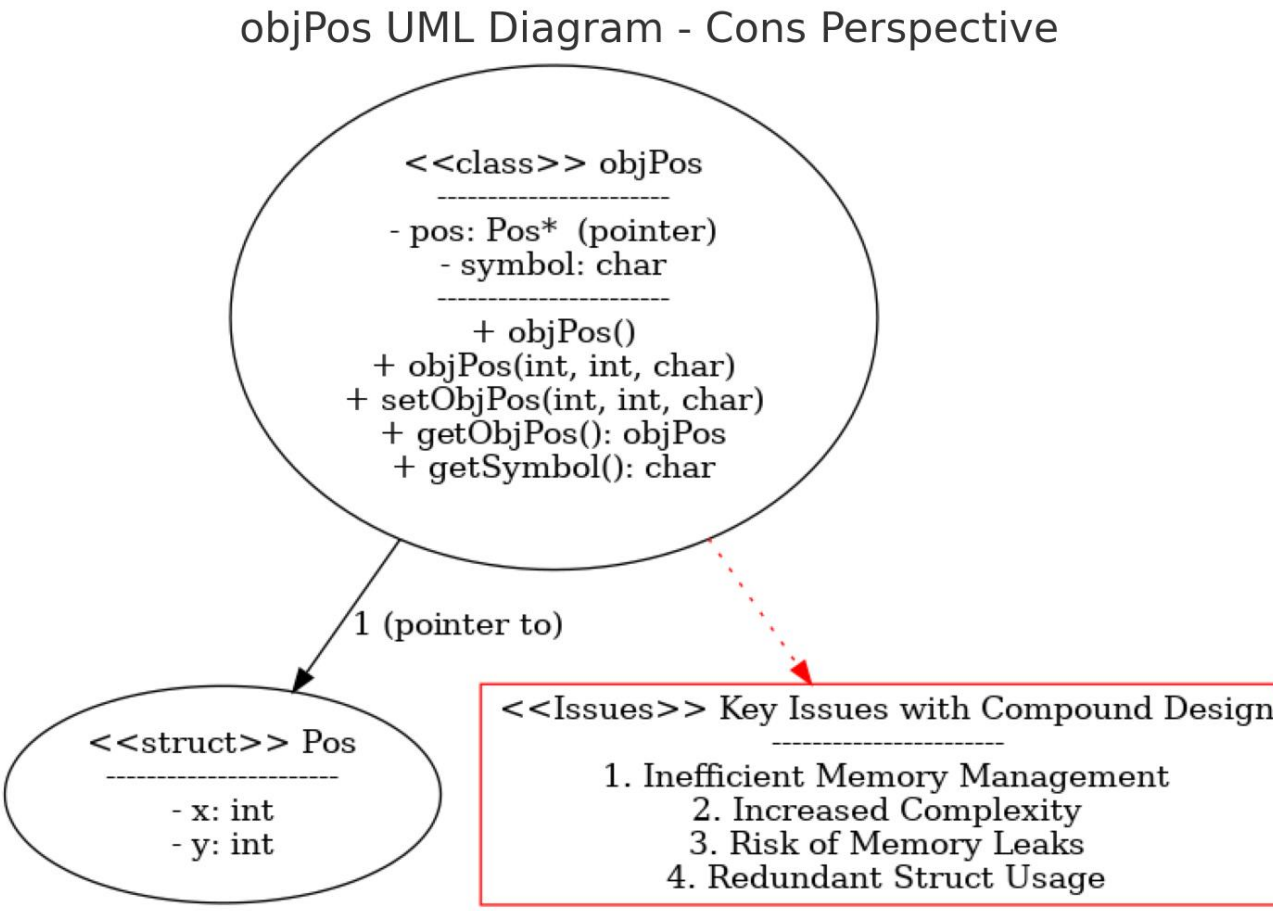
2. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.
   - **No memory leak**

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?
   - Overall, we do not think that it is sensible for the following reasons:
     - **Inefficient Memory Usage:**
       - The `objPos` class contains a pointer to a `Pos` structure, which requires dynamic memory allocation. This introduces unnecessary memory overhead and increases heap fragmentation.
     - **Unnecessary Complexity:**
       - Managing a pointer to the `Pos` structure complicates memory management. This could lead to bugs like memory leaks or dangling pointers if not handled properly.
     - **Redundant Structure:**
       - The `Pos` struct itself is simple, containing only `int x` and `int y`. These could be integrated directly as member variables of the `objPos` class, making the design cleaner and more efficient.

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.</u>
   - Merge `Pos` directly into the `objPos` class by using direct member variables (`int x, y`) instead of a pointer.
   - This simplification reduces memory overhead, makes the class easier to use, and eliminates unnecessary indirection.


objPos UML Diagram - Cons Perspective
<<class>> objPos
- pos: Pos* (pointer)
- symbol: char
+ objPos()
+ objPos(int, int, char)
+ setObjPos(int, int, char)
+ getObjPos(): objPos
+ getSymbol(): char
1 (pointer to)
<<struct>> Pos
- x: int
- y: int
<<Issues>> Key Issues with Compound Design
1. Inefficient Memory Management
2. Increased Complexity
3. Risk of Memory Leaks
4. Redundant Struct Usage