# COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

| | | |
|---|---|---|
| Your Team Members | Joud Al Mowesati | Vivian Le |
| Team Members Evaluated | Zimo Fu | Rui Zhang |

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.

Their RunLogic function seems to only to mainly use classes that they were provided, or they created themselves. It's very straightforward with the implementation, and what we really like about it is how neat and efficient the code looks, it overall deploys an object-oriented programming approach throughout the main logic loop.

This also however, though efficient, shows a lack of interaction between the classes deployed. You can't really see how they interact with one another, especially the processInput() class, as it isn't immediately clear how the user directly interacts with the main logic. Another thing to note is the game end condition they included, where one of the two conditions they compare the snake body size to an integer 200 which isn't immediately clear what the number entails. Defining a variable with a name and a size of 200 would be more straightforward regarding its use.

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:

- Separated classes for organization
- Encapsulation hides certain methods and data members from being modified by other classes
- The use of abstraction where the main program is simpler, without showing all the heavy lifting done by the other classes, the functions are only called when needed

Cons:

- More complicated, as the C procedural design approach is direct and straightforward, which is why the OOD approach was a bit harder to implement at the beginning.

## 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.

Their project offers sufficient comments in order to easily understand what is going on in each part of their code, and what every chunk of code is intended to do. Additionally, their use of descriptive variable names and functions make the code very understandable for parts that do not include comments. An improvement to increase the quality would be to add more detailed comments focused on "why" instead of "what" it accomplished.

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, their code showcases good indentation, sensible white spaces, and good newline deployment. Their formatting is very neat and easy to follow. They spaced out each function/ task, and with the combined comments, navigating their code was a breeze.

## 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 is smooth and has no apparent bugs after running and playing the game multiple times.

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, the Snake Game does not have any memory leakage as all members allocated on the heap memory has been properly deallocated with delete.

## Project Reflection

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?

Although unusual, the objPos class design with the addition Pos struct may be a memory-conserving strategy, this was not the most sensible design choice for the snake game. We say this because this overcomplicates the implementation of the game as we need follow the rule of 6/minimum rule of 4 for all the classes that contain an objPos type, where failing to do so will result in a memory leak. This design choice, although works, unnecessarily overcomplicates the program code.

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(s).

The current objPos class contains a pointer to a Pos struct containing x, y coordinates as well as a symbol. To improve the object design we could directly implement the struct with the x-y coordinates the same way symbol was done to reduce any complexity and make memory management simpler, instead of using a pointer (*Pos). If objPos directly used x and y coordinates, the class overall would become simpler too. As seen in the UML diagram, objPos contains methods like getSymbol, IsPosEqual, and getSymbolIfPosEqual. The number of methods could be reduced if x and y were handled directly, as the x and y values could be sent through getter, instead of as a struct pointer (*Pos), which would make the design overall more efficient. This UML diagram below represents the improved design of the objPos class that is more sensible to use for this project.