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

| | |
|---|---|
| Your Team Members | Alina Salam & Nida Siddiqui |
| Team Members Evaluated | Kristin Figueredo & Delayna Guenther |

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.

   Examining the main logic in the main program loop, it is easy to interpret how the objects interact with each other in the program logic through the code. The main loop is straightforward and follows a logical sequence: input, logic, rendering and delay. The program uses distinct functions that encapsulate specific tasks. Also, the objects interact cleanly with their scopes with responsibilities clearly divided. The DrawScreen function provides user-friendly prompts and visual feedback, improving the user experience.

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 of the C++ OOD approach includes encapsulation, abstract of dependencies, scalability, readability, and a foundation for future learning. Overall easier to upgrade aspects of the game.

   - Cons of the C++ OOD approach includes initial overhead (refactoring), tight coupling, and memory management complexity (potentials for memory leaks if not handles properly). Overall, more complex in terms of code.

## 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.

   The code offers comments throughout the different classes so the user can follow along with the creator's coding style. I was able to easily follow along with the code because the comments focus on the functionality of the code rather than stating the obvious details that can be seen just by looking at the statement. Overall, comments were used efficiently throughout the project.

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 follows good readability practices. There is sufficient indentation throughout each block of code making the overall structure easier to follow. White spaces are also used well between different methods and logic blocks, improving overall readability.

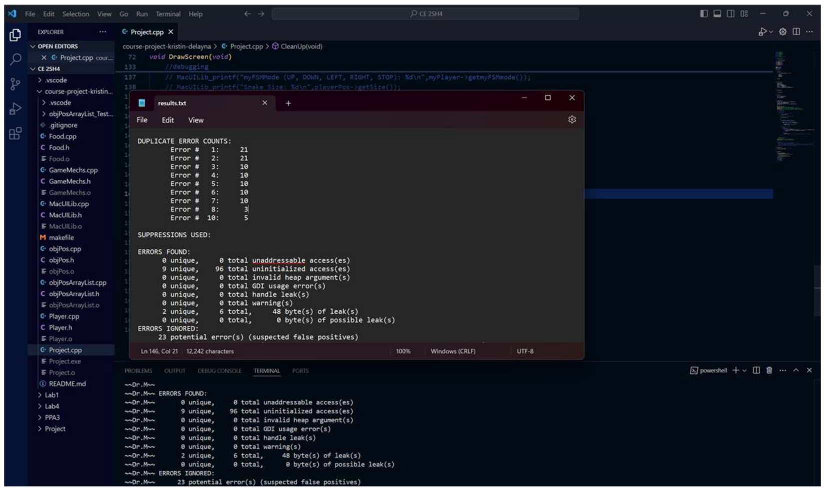
## Peer Code Review: Quick Functional Evaluation

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)

   Yes, the snake game offers an overall smooth and bug-free playing experience. I was able to go through several rounds of the game and grow the snake body quite a bit before eventually losing. The exit of the game is also fairly smooth as it deploys a message before the user is able to exit. It is also interesting that this team implemented a special feature that allows several points to be added to the score when the special food character "$" is eaten. Overall, the snake game offers a bug-free experience.

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.

   After using Drmemory, unfortunately, the Snake Game does cause 48 bytes of memory leakage. The report indicated 2 unique memory leaks which may root from small allocations such as an unused or uncleared objPos within playerPosList and/or leftover elements in foodBucket that are not properly deallocated during regeneration.



```
DUPLICATE ERROR COUNTS:
        Error #   1:     21
        Error #   2:     21
        Error #   3:     10
        Error #   4:     10
        Error #   5:     10
        Error #   6:     10
        Error #   7:     10
        Error #   8:      3
        Error #  10:      5

SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      9 unique,    96 total uninitialized access(es)
      0 unique,     0 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      2 unique,     6 total,     48 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
ERRORS IGNORED:
     23 potential error(s) (suspected false positives)
```

```
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       9 unique,    96 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total GDI usage error(s)
~~Dr.M~~       0 unique,     0 total handle leak(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       2 unique,     6 total,     48 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      23 potential error(s) (suspected false positives)
```

## Project Reflection

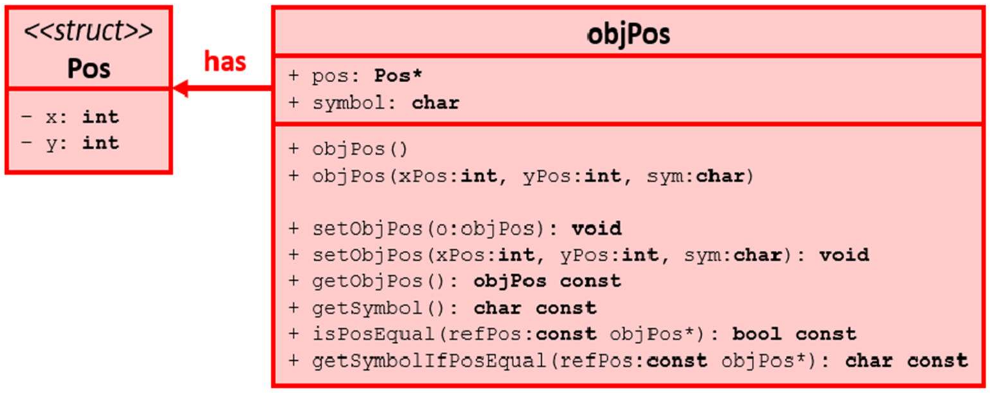
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

   The compound object design of the objPos class does not seem very sensible as it seems to make the code overcomplicated and a bit messy. Having to call the X and Y positions using the Pos struct operator every time seems redundant, and there could have been easier methods of accomplishing the same thing. This decreases the overall readability of the code and increases complexity.

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).

   An easier approach to solve the overcomplexity issue could be to use a member variable of type Pos and get rid of the Pos* pointer. As was the concern in the last question, this would make accessing the x and y positions much easier instead of constantly using the Pos struct. This also helps to increase code readability and makes the overall code less complex. The UML diagram is below.



<<struct>>
Pos
- x: int
- y: int
has
objPos
+ pos: Pos*
+ 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