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

Your Team Members Haseeb Shaikh, Tyler St-Amour

Team Members Evaluated (Mac Ids) mannic5, abbassj

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.
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 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?
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).

**Peer Code Review: OOD Quality**

1. Most of the code in the main program loop can be easily interpreted by how the objects interact with each other in the program logic, as most code is self-explanatory, and some comments are added to understand the logic of the code. Positive features in the code that we observed effectivelyly usess thethe different classes. Some negative features in the code include the need for commenting on some parts of the code to better better understand the logic. Additionally, the cleanup code could've been formatted better to allow us to read it more easily.

2. Pros:
   - OOD allows us to abstract away the minor details of our code, allowing us to only worry about how we interact with our classes (through member functions) rather than individual lines of code.
   - With OOD, we can easily organize our logic without needing one large file to do it all.
   - Since OOD code is modular, we can easily debug just one class or function rather than needing to test the entire application.
   - It allows us to share the workload since we can work in parallel on different classes instead of working one at a time on a single file.

   Cons:
   - For simple projects like PPA3, OOD can lead to more boilerplate code.
   - OOD requires more thoughtful planning to structure your project.

**Peer Code Review: Code Quality**

1. Some parts of the code need to offer sufficient commenting, making it challenging to understand the logic. For instance, in the Food class, they created multiple objects for randomization purposes without commenting on their use, and their use cannot be easily interpreted through the code. Additionally, more comments could be added to the Project.cpp file to comment on the use of invoking different functions.

1. The code mostly follows good indentation, use of spaces and newline formatting. However, in some areas of code, it falls short. For example, in the cleanup and generateFood functions. To fix this, they should add some lines between large code blocks and ensure correct indentation across all files (or use the VSCode auto-formatter). Additionally, there are some improper use of spaces in the output, and more lines should have been added instead of printing all the features in one line, such as score and instructions to above and beyond features.

**Peer Code Review: Quick Functional Evaluation**

1. The Snake Game offers a smooth experience, but it contains a bug where the food can be generated on top of the snake. This is caused by line 32 of Food.cpp, where y values are incorrectly checked to be x values. This could be solved by checking the x and y values correctly

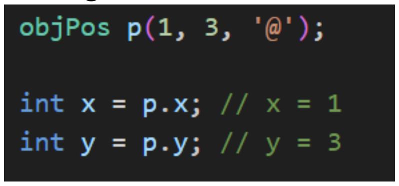
or using the isPosEqual function of objPos. Some other improvements that could be made are using a bigger game board than PPA 2 (which we aren't sure if this was a requirement), showing the user the game board after they ended or lost the game instead of just printing the message, better choice of symbols as it makes it confusing for the user, and increasing/decreasing the game speed to make the game more interactive.

2. The Snake Game does not cause any memory leaks after running drmemory to check for any memory leaks in the code. It can also be seen through the code that the group effectively allocated and deallocated memory on the heap.

## Project Reflection

1. We think that the use of composition in objPos was unnecessary and does not make the programmer's life easier. Since objPos is a class intended to store an object's position (and symbol), it doesn't make sense to use composition to store a position. The objPos class should have position functionality implemented natively in the class. However, if Pos was a class that provided some complex functionality, then an argument could be made for composition. For example, if our game was 3D, Pos could have been a class that stores x,y, and z and provides member functions to calculate the distance to another point, calculate unit vectors, etc. Overall, objPos was not a necessary case for composition.

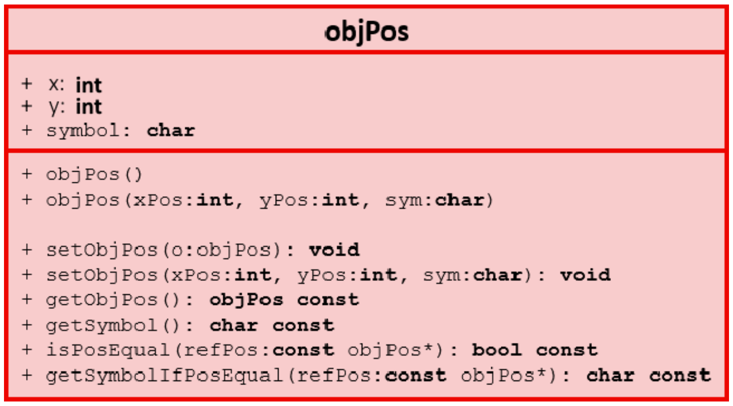
2. We can improve the object design by using x and y as public fields rather than using a struct to refer to the x and y values of the object. This will improve our design by using less memory and making the code shorter. So code using this new objPos would resemble this:



```
objPos p(1, 3, '@');

int x = p.x; // x = 1
int y = p.y; // y = 3
```

The UML diagram for the modified objPos class would look like this:



```
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
```