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

Your Team Members alsamml seetk

Team Members Evaluated kaileg2 asilara

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.

   From the main logic, we are able to interpret generally how the objects interact with each other. We can see how the player interacts in the game through method names which are called from the object pointer "myPlayer" which include movePlayer and updatePlayerDir (whose names indicate the actions done through these methods). Stepping into these methods, we can further interpret how objects across the project interact. In the updatePlayer method, we can see the interactions between mainGameMechsRef with the player which is necessary to relate input to player direction. Next, in the movePlayer method, we can explicitly see the interactions between the player and food to detect collisions and understand the collision mechanism of the game. Therefore, we can easily interpret the necessary object interactions by looking at the main logic and understand the game at a high level, making this a positive feature.

   However, the main logic does not provide underlying relationships between objects that are hidden within other method implementations. This can be a positive and negative feature as it can safeguard information you don't want external programmers to see but also might hide relatively important object interactions that would aid in their understanding.

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

- Modularization prevents unintended interactions between different parts of the program.
- Encapsulation helps to protect data by restricting direct access. In the procedural design, global variables are accessible by all functions which could lead to unintended modifications.
- Class methods can be reused and help reduce redundancy / duplication of code.
- Improves code structure and organization, greater readability can be achieved through method naming, holding more complex code.
- Supports scalability without disrupting existing code.

- Easier to locate and debug issues with obvious root causes when code is grouped into a method.

Cons

- Features of OOD such as object creation and dynamic memory allocation can introduce runtime overhead.
- Can become a hassle looking through classes and methods to review a code segment vs having all/most of the logic/code in the main program file.
- Harder to debug issues with unknown root causes (have to go through all classes).

## 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 provides thorough comments that explain the functionality of code blocks. The comments are descriptive and avoid redundancy, by explaining why actions are performed rather than just what the code does. For instance, "Shifting snake right by 1 character" when describing incrementing the x coordinate of the currentHead. The comments are used for explaining more complex logic and not for trivial lines of code. The variables and methods they created have clear, descriptive names (ex. tempFood, setFoodIndex, thisGMRef, etc.).

   However, there remain some comments from their debugging process that are unnecessary or misleading, which should be removed to improve conciseness. For example, unnecessary preprocessor directives and outdated code logic.

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.

   The code does follow good indentation practices and sensible white spaces. The code includes indentations in if statements, switch cases and loops for better readability and organization. Code segments appear to be grouped according to use/function through using newlines and whitespaces; for example, initialization of variables and logic are separated by spaces for greater organization and readability. Game messages employ newlines to promote better readability for the user.

   A shortcoming observed is that there are some inconsistencies in the use of curly brackets; sometimes they are used on the same line as an if/for statement while other times they are on a separate line. Keeping the indentation and braces consistent throughout the program would enhance 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)

The Snake Game offers a smooth, bug free playing experience for the most part. The snake moves smoothly using WASD control, detects food collisions and randomly generates new food after collisions. The score increments correctly and the game exits when the snake collides with itself.

However, when changing the speed of the snake, it is not very clear whether the speed has changed or when it has reached its maximum/minimum levels. To overcome this and create a smoother experience, the team could print a speed mode message that changes when speed is increased/decreased. Furthermore, one bug observed was when 2 or more food items were generated in the same row, the border pixel (#) of the same row shifts to the right. Using previous programming language, the possible root cause is located in the drawScreen function, specifically the drawing loop. This is because it appears to be a printing/output issue rather a logic issue. An issue in the for loop of drawing the game board with all the features is likely to cause this bug. The debugging approaches the team can deploy is using debugging tools like the IDE debugger or print statements (in this case print statements would work better). I recommend adding print statements once an item is printed. This print statement can output the symbol printed as well as the location (i and j). These print statements can identify where bugs are occurring if the location is out of bound which can be traced back to the code and help identify the issue.

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.

Yes the snake game does cause memory leak, as seen in the image below.

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
     11 unique,   131 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)
      6 unique,  9964 total,  79712 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)
         (details: C:\Users\kasee\Downloads\DrMemory-Windows-2.6.0 (3)\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
project.exe.38872.000\potential_errors.txt)
     35 unique,    36 total,   8284 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
  31122 leak(s) beyond -report_leak_max
Details: C:\Users\kasee\Downloads\DrMemory-Windows-2.6.0 (3)\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-project.exe.38872.000
\results.txt
```

However, the team did delete all memory from the heap and included destructors which also deallocated heap members initialized in constructors using the "delete" method (number of new and delete is the same). Therefore, I think the possible root causes are not from missing deallocating heap members, rather it could be an error of the program which is possible as mentioned in class before and was applicable to our PPAs.

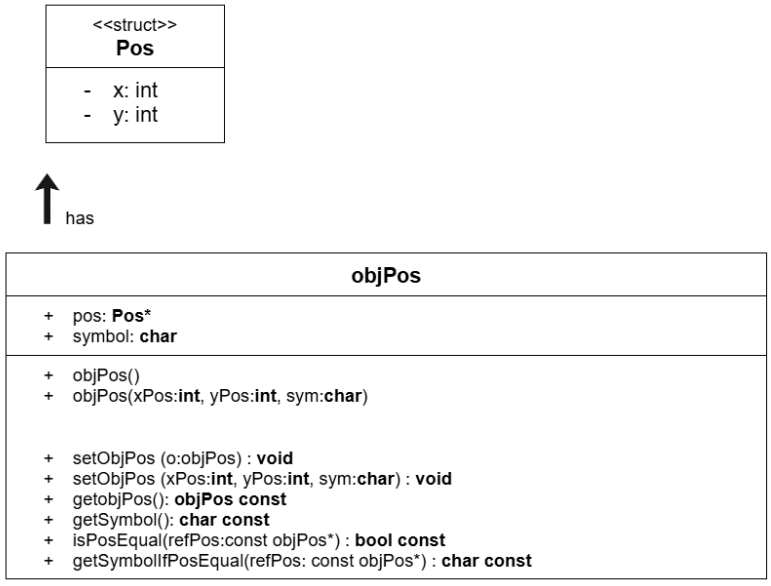
## Project Reflection

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:

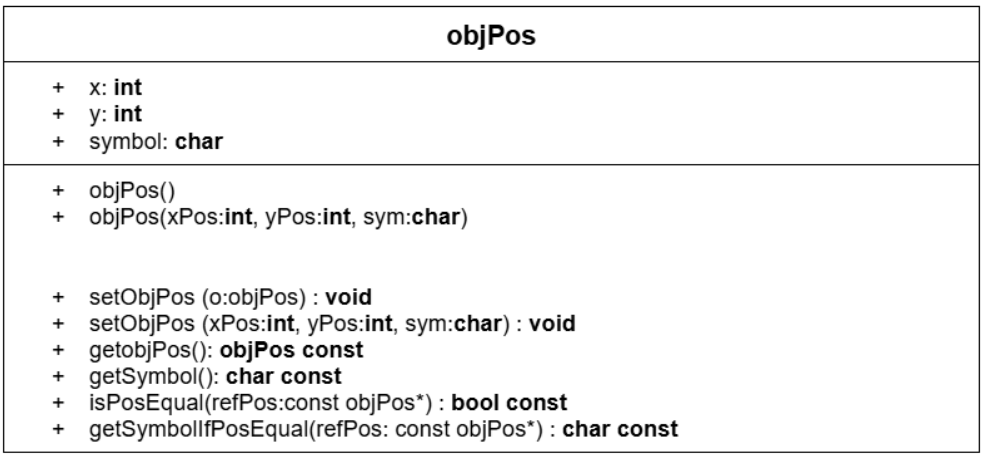
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?
2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. <u>You are expected to facilitate the discussion with UML diagram(s).</u>

1. I do not think the compound design of objPos class is sensible. I think it makes the code less readable and more complex, especially since an x, y, and symbol make up an object, just like how the special constructor in the objPos class include x, y, and symbol as its parameters to initialize the object. Using the additional Pos struct required the pos pointer to be a part of the class, which made the code more complex due to required referencing to obtain x and y positions. For example, when accessing a food position, this is how it was obtained due to use of a pointer to the Pos struct:
   FoodPos.pos->x and FoodPos.pos->y
   While accessing the symbol, which is part of the objPos class, was simply done by this:
   FoodPos.symbol.
   Therefore, I do not think the design of the objPos class was sensible, especially since symbol was made a member of the objPos class but x and y were made a separate struct and referenced.

2. An alternative objPos class design that is counterintuitive would be to include x and y variables in the public section of the objPos class, just like the symbol. This way, the struct would not be needed to instantiate objects, and more importantly, a pointer to the Pos struct is not needed as well as dereferencing to get the x and y positions. Furthermore, public members means easier access as opposed to continuously using a getter function, which would be tedious due to the number of times it would have to be called.
   The current UML diagram for the objPos class looks like this:




Alternative class design:




This class design is without the additional Pos struct, avoiding unnecessary complexities and making the class more intuitive. It is more evident that an object comprises an x position, y position and a symbol.