COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members	khehrn2, sasits1 (khehrn2-sasits1)
Team Members Evaluated	maharv3, pirament (free-p)

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.

   The main logic in the main program loop follows a clear and easy-to-understand structure. The Player, Food, and GameMechs objects were clearly defined and used appropriately in the program; each class handled its own specific logic, making the program simpler. The objects demonstrated their specific functionality and I could easily interpret the interactions between them throughout the program code. Their main code was also structured in a flexible manner, making it easier to build off of or extend their program to more functionalities and features. The code would benefit from some more documentation to help readers understand key object interactions. Additionally, some variable names like "sign" could be renamed to more descriptive alternatives that reflect the functionality of the variable and help readers understand the code better.

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.

   <u>Pros of C++ OOD Approach</u>
   - Classes and objects can be modified easily and reused in other projects
   - Can build upon the current project and add advanced features easily – ex. food -> foodBucket
   - Better readability of the interactions between classes and objects
   - Each class has a specific objective, making the code easier to understand and maintain

   <u>Cons of C++ OOD Approach</u>
   - Classes and objects may make the code more complex
   - DMA also introduces some issues in code
     - Increased possibility of memory leakage due to allocating memory on heap – ex: new, delete

   <u>Pros of C Procedural Design Approach</u>
   - Code is easy to understand and follow through functions
     - Functions can be reused multiple time, reducing repetition and increasing readability

    - Independent functions can be tested before being used in main code, reducing time and effort for debugging
- Demonstrates a consecutive flow making it simple and clear to understand

Cons of C Procedural Design Approach
- The use of global variables in procedural design can make it more difficult to identify to the root of the errors in the code
- Procedural code is hard to reuse and not very flexible
    - Functions are too specific to a certain context, that it becomes difficult to extend their functionality without modifying the code
- With bigger projects, the code becomes hard to navigate due to limited modularity

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 a reasonable level of documentation, specifically in the Player classes, where they explain the reason behind specific operations like moving the player. Most of their method names and variable names were clear and described the functionality accurately. However, some variable names like “sign” to describe the segment checking, were unclear without context. Areas like the generateFood function in the Food class that would have benefitted from more documentation behind the code functionality as well. Their collision check code in the Project file would also have benefitted from commentary. Overall, the code was relatively easy to understand through the documentation and provided good readability.

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 provides sufficient indentation and use of newline formatting to separate the different code blocks. Additionally, white spaces were added between separate statements to enhance the code quality such as the generateFood function in the Food class and input processing logic as well as FSM in the updatePlayerDir function in the Player class. To improve the readability of the code, it would be beneficial to align the opening and closing braces within in the blocks of the if-else statements of the Player.cpp, objPosArrayList.cpp, and objPos.cpp files to better indicate the block of codes to be executed with each corresponding conditional statement.

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 provides a smooth, bug-free playing experience with an appropriately selected board size and character symbols chosen for the player as well as the food object. Additionally, when a collision occurs between the snake and the food object, a new symbol is immediately added to the head and boundary wraparounds are obeyed. To improve, it would benefit the playing experience if the commands (e.g. w, a, s, d) were printed to the terminal to indicate how the movement of the player works and correctly spelling "score" in the terminal (in the place of "socre"). Overall, the game runs smoothly but could improve with terminal instructions to describe the player movements.

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 running the executable and reviewing the memory profiling report, the Snake Game does not produce any bytes of leak that contribute to memory leakage. The memory profiling report indicates 0 bytes of leak(s) and 0 byte(s) of possible leak(s). The program properly allocates memory for heap members using the keyword "new" and deallocates the heap members in the appropriate destructor using the keyword "delete."

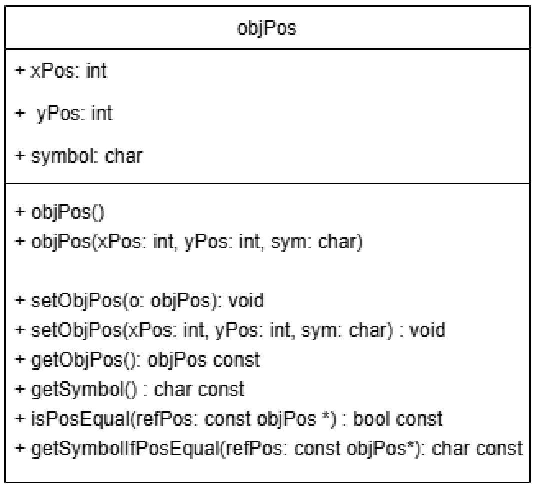
<u>**Project Reflection**</u>

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:

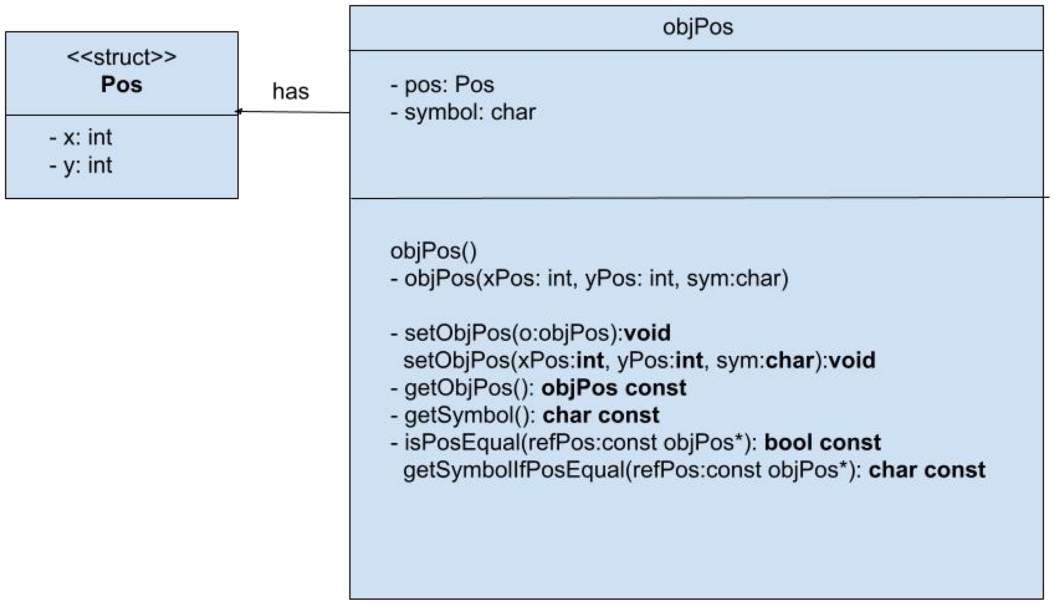
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?
   The compound object design of objPos class is not highly sensible as it uses the pointer Pos* pos to dynamically allocate memory for the two variable members (int x and int y) in the struct. This pointer unnecessarily makes the programmer have to manage memory the heap using the keywords "new" and "delete". Additionally, the setObjPos method of the objPos class passes an objPos object by value which requires the Pos* pos pointer to be copied as well. The function calls for setObjPos would result in slower execution due to the objects being pass by value (copies of the object values are modified) whereas passing by reference would be more efficient (original values of the object are modified).

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>

   One alternative design for the objPos class is to eliminate the Pos struct and have the objPos class directly store the x, y, and symbol variables in the objPos class. This simplifies the memory management needed; we no longer need to allocate or deallocate memory dynamically, which reduces the risk of memory leakage. Accessing the variables directly would also be faster and result in better performance of the program. The constructor in objPos would directly store the Pos object: pos with initial values of x and y to be zero with the following: objPos() : pos{0, 0} {}.


objPos
+ xPos: int
+ yPos: 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

Another alternative design would be to keep x and y in the Pos struct, and use it as a member in the objPos class. In this design, Pos is not a pointer, it is an object so there is no memory allocation needed here either because the object is created on the stack; so the object is destroyed automatically. This eliminates the need for pointers in the objPos constructors and methods, and gets rid of the "new" and "delete" calls in program. The x and y variables are grouped together and can be accessed directly from the Pos struct, making the logic of the program easier to understand.


<<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