COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members:Team Name: LScoding Names: Shaya Jabbarzadeh(jabbarzs), Licheng Zhou(zhoul103)___

Team Members Evaluated: Team Name: asim-alsoofi MacID's: asimh6, alsoofin_

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.

   By looking at the main logic in the main program loop in the Project.cpp file, the program is structured in a clear and organized manner. This makes it easy for me to understand each line of the code I am reading since the name of the functions they call provide enough information to have an idea how what is happening at each step. Although there is no issues, it is possible to improve the DrawScreen() function by first checking if the map is on the border to draw the border of the map first. This is because you can then use an else if statement to check if the food needs to be rendered in that position, followed by an else statement containing the code to render the player. This is because if you can confirm (without the process of looping) that the current tile to render is either the border or the food, you can avoid an unnecessary loop of the player. This change, although not necessary, can be done to optimize the game as it reduces the number of times your program will loop through the player list.

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.

   The pros of the C++ OOD approach is that all the code can be organized into a single object which provides us the ability to make cleaner code which is more readable and accessible. However the cons of C++ OOD is that it is slightly harder to understand how memory allocation and deallocation works.

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 does provide sufficient comments with a consistent coding style, making it easier to tell the difference between a variable and a function. There is also very good practice with variable naming as the name of every variable correlates with what their purpose is.

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 and formatting as everything is uniform throughout the code. The only potential shortcoming is that some of the lines of code move horizontally rather than vertically, so it would be good to add new lines in very long lines to make sure the reader does not need to use horizontal scroll.

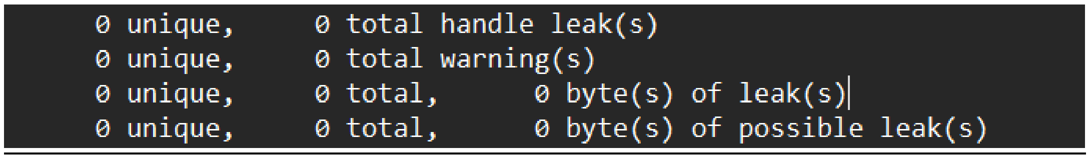
Peer Code Review: Quick Functional Evaluation

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)

   As far as my testing as gone, I have not found any problems with the playing experience. From my own personal experience, if you are ever having any issues with your program, the best thing to do is use the debugger from the start of the main loop and to go step by step to see if there is anything wrong using watch variables. For example, if you created player object and pass a gameMech object that isn't initialize, the program would immediately crash. By going step by step, you can see where you made these mistakes easier. Another tip is to use Memory or other memory debugging utilities to see if your program is making any problematic memory operations, like issues that arise from missing a copy constructor and copy assignment constructor (Not just memory leaks).

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 Dr Memory and playing the game, at the end there were no memory leaks detected.



```
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      0 unique,     0 total,      0 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
```

Project Reflection

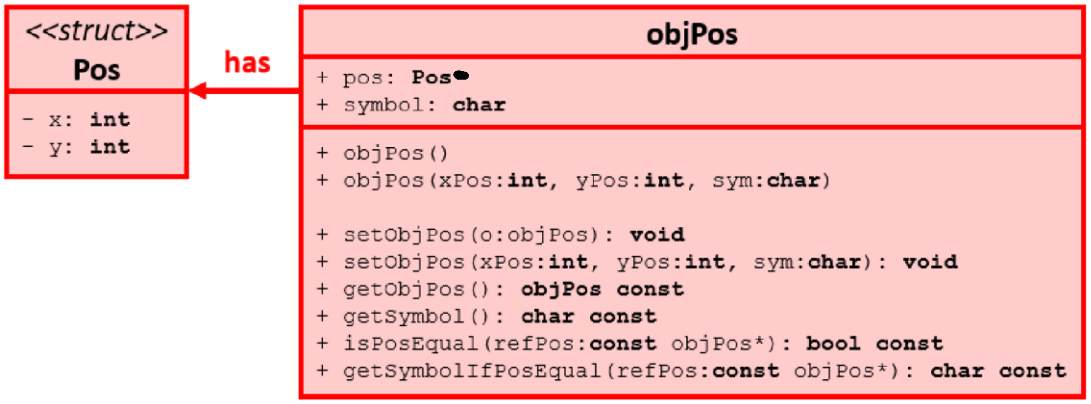
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

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

   The compound object design of objPos class is sensible as it allows for other classes to use the position of x and y making it modular. This is extremely important as you can easily connect each object with their respective positions (x,y). For example, when detecting collisions, the use of the object position class with the additional Pos struct helps to distinguish between symbol and position. Our implementation and the other team's implementation are relatively similar. However, making a pointer point to the struct is not as efficient as you are risking memory leaks.

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>

   An alternative objPos class desgin that I believe is relatively counterintuitive than the one in this project would be to replace the Pos* pointer with an instance of Pos pos in objPos. This eliminates the need for dynamic memory allocation. By making the struct object a value, this allows accessing position data much faster as you do not need to dereference to get x and y positions like we did in our project.


<<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

   In this UML diagram, the instance of the Pos struct is not a pointer anymore, but instead a direct object which makes life easier when having to call the x and y positions from objPos.