COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members	Alexander Gascho (gaschoa), Joey McIntyre (mcintj35)
Team Members Evaluated	huqm6, syedh42

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 **30 marks**. Do not exceed 2 paragraphs per question.

<u>Peer Code Review: OOD Quality</u>

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.

<u>Peer Code Review: Code Quality</u>

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.

<u>Peer Code Review: Quick Functional Evaluation</u>

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.

<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 yours, 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.</u>

Peer Code Review: OOD Quality

1. Their main logic within their main program loop in project.cpp is well structured with intuitive design. The different responsibilities, such as input handling, game logic, and screen drawing, were clearly separated and distinguishable from one another. Their code is also written such that the interaction between the Player, GameMechs, and Food objects is clear. Overall, nothing stands out as a major issue in their main program loop. One slight improvement they could make is to remove some redundant logic checks. For example, in DrawScreen when they check lose and exit flags they check "if (myGM -> getLoseFlagStatus() == true)" which could be simplified by removing the "==true".

2. Pros:

    - OOD allows us to organize the program so that responsibilities (i.e RunLogic, Initialization, etc.) are clearly separated and organized
    - The modularity allows issues/bugs in the program to be narrowed down to specific classes, member functions, and/or variables for debugging
    - Using OOD allows the program to be improved and added to with ease
    - Multiple classes allows teams of programmers (in this case 2) to work in parallel with minimal risk of conflicting code

    Cons:

    - OOD requires significantly more planning and effort to setup which might not be worth it for a small project like a snake game with a clear end target (does not need to be maintained/expanded on after completion)
    - Also, compared to PPA3 the OOD approach requires significantly more memory management throughout the program

Peer Code Review: Code Quality

1. For the most part, their code was intuitive and written in a sufficient self-documenting style. Most of their logic is easy to interpret with adequate comments to clarify more confusing sections. They could improve their overall readability of their for loops by choosing more descriptive variables. For example, instead of using "i" and "j", they could use "row" and "col" or "x" and "y". But this isn't a major issue since they had adequate comments to explain each variable.

2. Overall, their use of whitespace, indentation, and newline formatting was excellent. Throughout the project they made sure to separate their variable declarations/initializations from other blocks of code. They also ensured to break up larger blocks of code, such as their switch statements in MovePlayer, with either whitespace or comments.

<u>Peer Code Review: Quick Functional Evaluation</u>

1. Their game is a smooth experience with no bugs observed in our testing. There is one minor flaw in their food generation which doesn't impact the current implementation of the game but does prevent their board size from scaling. Currently, in GenerateFood in their Food class, a new instance of GameMechs is created on the stack to derive the board size. If GameMechs object in the main program loop was created with a custom size, GenerateFood would still generate based on the default board size. This could be fixed by passing a GameMechs object into the Food constructor.

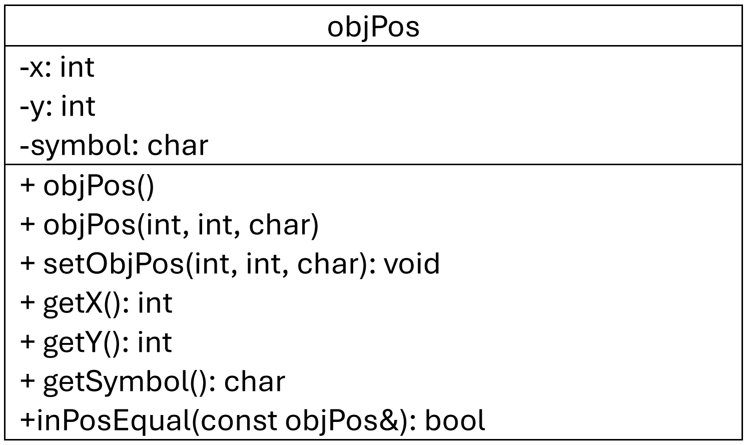
2. Their code adequately deletes/deallocates all allocated memory throughout their program. This was confirme Iteration 0: d by a drmemory report which stated that there was no memory leak.

<u>Project Reflection</u>

1. After reviewing the other team and our own implementation, we decided that the compound object design of ObjPos is not sensible. Since the purpose of the ObjPos is exclusively to store the x-y positions and symbol of an object, using a struct within the class for position doesn't make sense. This method adds unnecessary complexity and risk by instantiating Pos on the heap. However, compound object design could be a benefit if Pos was a class which contained additional member functions, or varied in dimension (2D or 3D).

2. The compound objective design of objPos is not ideal, as it includes a number of unnecessary implementations that impact the codes performance. We could improve the design of objPos by removing the struct Pos, and adding x, y and symbol as members of the class. This is a simpler way of achieving the same end result, that also improves performance due to less dynamic memory allocation. This would also make the peer review evaluation process easier as the code would be more logical and easier to follow.

This improved object design can be supported by a UML diagram of the changes. So this is what a UML diagram for the improved design of the objPos class.



This is the UML diagram. As you can see, Pos is removed entirely, and x, y, and symbol are added as members of the class.