COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members: Jordan Stepak (stepakJ), Vanessa Harvey (harvev1)

Team Members Evaluated: Caden Chan (chanc167), Tyler Fong (fongt5)

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 teams code is relatively easy to interpret and understand how they made the different classes and object interact. They have a good number of comments outlining what each section of the code does, making it easier to understand the logic and more complex parts of their design. As far as positive features go, their check food consumption logic is very interesting in that they return the actual character of the food they are consuming so that they can access it via the main project logic rather than within the player class. One feature that could be updated, is making their if statement in the run logic section a switch case. It's a major change, but since they are already checking a specific character that's returned it does make it a bit more simplistic and makes it easier to add additional food items in the future.

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 OOD approach helps make the code more modularized.
   - It allows for additional features to be added based off the same behaviour without directly copying hundreds of lines of code multiple times.
   - The OOD design makes it much easier to add additional features to an already working game, rather than starting from the base and working up all over again.
   - The OOD design makes passing object specific values between different classes more difficult, as not all classes have access to each others' features/members. This is both a pro and con, allowing for better modularity and less possibilities for error, but more difficult when programming object interactions between classes.
   - The OOD design makes it significantly more difficult to track down bugs or errors that a buried deep in a specific class, that may not show themselves until you call a specific function. This makes it much more difficult to determine where the bug is actually taking place. We experienced this ourselves.

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 offer sufficient comments in the main code files such as the player, project, and food. However, the comments in the other files are a bit limited. In the main files, the comments do explain the most important aspects of the code and help to have anyone reading it properly understand what each section does. As for the other files, having some more additional comments to simply label what each member does and how it contributes to the overall code would improve the understanding. Especially when it comes to debugging, if a new user needs to track what each member does and see where it goes and what it calls.

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 has good readability, utilizing white spaces to break up long chunks of code, proper indentation and newlines/spaces within printed outputs. This made playing the game an intuitive experience and following/understanding their code easy.

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)

   There do not seem to be any bugs in the game. The game runs smoothly and properly, giving the player the proper number of points for each food, and increasing the length properly. Additionally, the wraparound logic works correctly, as well as the escape and lose functionality.

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.

   The snake game causes 76448 bytes of memory leak. The root cause of this memory leak is from the project.cpp file in the clean-up routine. At the beginning of the main program logic in project.cpp, three elements are initialized on the heap, "game", "food", and "player". In the clean-up routine, none of these elements are deleted from the heap, causing this memory leak.

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      8 unique,    93 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)
     12 unique,  9556 total,  76448 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 the objPos class is useful in some sections of the program to compare object positions, however it overcomplicates the objPos class. In multiple instances, the x and y coordinates of an object are needed separately to compare with positions on the game board, and the use of the Pos struct makes obtaining these separate integers unnecessarily complicated. To compare these two sets of coordinates it is necessary to create new objects containing the board coordinates, making the program less memory inefficient.

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).

   An alternative objPos class design would include separate x and y members rather than a Pos struct containing the x and y coordinates. This would include separate getters and setters for x and y, making it easier to compare with coordinates not held in Pos structs. This design would still contain member functions comparing the x and y positions at once for ease of use, which would increase the overhead for building the class but make the use of the objects easier.

ObjPos
+ x: **int** + y: **int** + symbol: **char**
+ objPos() | objPos (xPos:**int**, yPos:**int**, sym:**char**) + setXPos(xPos:int): **int** + setYPos(yPos:int): **int** + setSymbol(sym:char): **char** + setObjPos(xPos:int, yPos:y): **void** + getXPos(): **int const** + getYPos(): **int const** + getSymbol(): **char const** + isPosEqual(refPOs:**const** objPos*): **bool const** + getSymbolIfPosEqual(refPos:**const** objPos*): **char const**