COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members Mya Spoelstra + Peter Jabra

Team Members Evaluated ~~MacUILib_init();~~ Our assigned team, MacULib_int(), did not complete majority of their code thus as per Dr. Arthar's instructions, we will be evaluating and analyzing our own code.

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.

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.

   Yes, the main program loop is easy to interpret, and it is clear how the Player class interacts with GameMechs and Food class. With the use of helper functions, make it easier to understand how the Player handles movement, delay, and speed, while GameMechs manages score and input, and Food generates food. The design is clear, but the interactions in DrawScreen() could be made more explicit for better clarity. For example, the logic for how the Player and Food detect collisions can be extracted into helper functions, making it clearer when and how the two objects interact. And the logic for how the board is created could also be moved into separate functions in order to make when and how the board is generated clearer. This would improve readability and make the code more modular, ensuring that each part of the game logic is more clearly defined.

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:
   - OOD is ideal for **reusability** as its design promotes code reuse and reduces redundancy through inheritance
   - Allows for **better organization** and separation of concerns through class separation
   - **Easier to work in parallel** as team members can work on different classes

   Cons
   - OOD has **more complex** structure which can make troubleshooting and debugging more difficult
   - Abstraction and OOP features (like classes and object creation) can **make the program slower** and use more memory

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.

   Yes, the code offers sufficient comments and is organized in an easy-to-read format. Each function is clearly marked with comments explaining its purpose and what the code is doing. For example, in Food.cpp, lines 22-87 are well-commented, with a clear explanation at the beginning of each function about its functionality. This makes it easy for someone reading the code to understand what each part does. The comments are concise and relevant, which helps maintain clarity throughout the code. The only part of the code that could benefit from additional comments is in Player.cpp from lines 273 to 312. While it's noted that these functions handle regular movement and check for collisions, adding a few more comments within each function would improve the readability and provide further insight into how the functions work. This would make it easier for others to understand the logic and potentially troubleshoot any issues that arise.

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 follows good indentation, sensible whitespace, and clear newline formatting throughout the entirety of the code. Appropriate spacing between the different functions within each class is consistent across all the classes, and the code is neither cramped nor over-spaced. It is easy to follow the different functions as they have a clear start and stop, thanks to the spacing. If the syntax of C were unknown, the spaces alone could guide the reader in identifying which functions are together or separate.

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 code is free of bugs and runs smoothly without any noticeable issues. However, if the code did run with bugs, we would recommend using the testing suite in order to iron out the flaws. This is because the testing suite given to us is through and universal (every team should pass it). By following and making sure your code passes the test cases outlined in the testing suite, you can easily set the rest of your project up for success. If memory leakage specifically is found, our team recommends, doing a search for the word "new" in all files, and as you do so, ensuring that the word "delete" can be found alongside it somewhere else in your program. In our opinion it is extremely easy to mix memory deallocation, and an extremely frustrating issue to try and fix, so this is the method our team utilizes every time we have an issue, in order to perform a through sweep of our code.

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.

   There was no recorded memory leaks in the report generated.

Project Reflection

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?

   We think having the compound object design of objPos is slightly unintuitive. This is because now the programmer has to go through another extra variable to access the x and y values of the objPos.

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.

   A simpler and more intuitive approach would be to store x and y as direct member variables within the objPos class, without nesting them in a separate Pos struct. This would allow for direct access to the position, making the code more straightforward and easier to follow. By doing this, the position data becomes a natural part of the objPos class and would be more intuitive to access. There would be no need to navigate through a nested structure, which would reduce complexity and improve clarity.

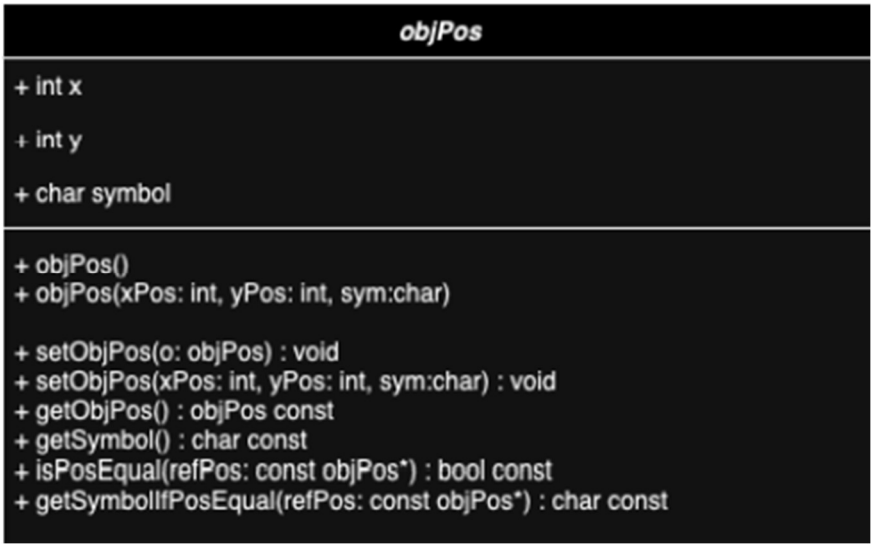

Figure 1: UML diagram for alternative Pos Struct