COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members Hasan Asim and Nooraldeen Al-Soofi

Team Members Evaluated zhoul103 and jabbarzs

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.
   When looking at the main program loop, we can easily interpret how the objects interact with each other in the program logic. For example, we could easily see how the *Player* object handles movement and direction, *GameMechs* manages game states and input, and *FoodBin* generates and tracks food's position. The RunLogic function ties these objects together, showing how they work as a team to make the game run smoothly. However, something we would improve moving forward is the interaction between myPlayer and the foodBin. When looking at movePlayer we assume it looks for food collisions, but it isn't clear how the player movements affect this state. This made it a little harder to interpret the code in terms of understanding how the players movement works with food collection process. Generally, the program is easily interpretable, but adding clearer object interaction between myPlayer and the foodbin would make it even easier to understand.

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.
   OOD Pros and Cons compared to Procedural design

Pro's	Cons
- Classes and objects can be easily reused throughout the project. - By using abstraction, we can hide implementation details, which in turn allows the code to be easier to be read and understood - New features can be easily implemented, making the program much easier to scale compared to procedural design approaches, thanks to inheritance and polymorphism. - Reduction of errors and bugs as we can avoid the risk of global variable misuse.	- Procedural design approach is simpler compared to C++ OOD as usually with OOD the complexity increases. - Dynamic memory allocation is more difficult in OOD, making it more prone to potential memory leaks compared to procedural design. - Debugging is more difficult in OOD because the logic is spread across multiple classes and functions.

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.

   Although we do understand the structure of the code, if the code was commented more, it would enhance readability and make the functionality clearer. There are some basic comments in place, however, it lacks them around more complex areas in the code. This way the code's logic is easily understandable without needing to 'decipher' as much. A way this could be implemented is by adding more inline comments for complex classes and their interactions. We believe this would greatly improve the code's comprehension.

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

   When looking at the code, it is very well indented, making it easy to understand the flow of execution and helping to identify which code belongs to which block or scope. If we could change one thing, it would be to add more spacing or blank lines to separate different logical sections. This would help avoid tightly packed code and make it much easier to navigate.

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 runs smoothly and works well based on my testing. The game loop behaves as expected. The player movement works well as every possible input results in the correct action, eating food extends the length of the snake and adds score accordingly, as well as the detecting collisions works as it ends the game immediately. The sudden flickers on the screen can be bothering whenever the screen is constantly cleared with MacUILib_clearScreen() however it isn't much of a mentionable issue.

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 were not any memory leaks as all the object pointers were deleted in the clean up as well as in their destructors.

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?

Yes, the design is sensible because it groups the position (x, y) and symbol into a class, making the code easier to understand and keep organized. But I feel using dynamic memory for x and y is unnecessary and complicates the code. It causes potential risks of memory leaks and pointer issues, which can lead to bugs.

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 counterintuitive design would be to combine x, y, and symbol into a single array. For example, using an array like int data[3] to store these values would make the code harder to read and manage. It would require you to remember that data[0] is x, data[1] is y, and data[2] is symbol, which is confusing and not intuitive. To improve the design, I would remove the dynamic memory allocation and just store x, y, and symbol as regular class members. This will eliminate unnecessary complexity. I'd also add getter and setter methods for easy access to these values, which would make the class simpler to use.