COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members Jyotiraditya Pendyala, Andrew Chai

Team Members Evaluated

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.

   The logic in the main program loop is modular and easy to follow while adhering to the object oriented approach. The variable names are also very intuitive and can be understood easily. The less intuitive lines of code such as print statements for special symbols are explained using comments making the code readable and easy to interpret.

   There are, however, a few points of criticism. First, while the delay constant was defined as 50000 µs, the loop delay function has a comment mentioning 0.1s. This may cause some confusion to people unfamiliar with the code. Second, in the CleanUp() function, the pointers called on heap are deleted directly instead of calling in their respective classes' destructors (this is more procedural than object oriented). In addition, food consumption and exit key checks are occurring within the RunLogic() function instead of their classes' member functions, once again breaking the modularity of object oriented code and reverting to procedural. Lastly, from a player's perspective, the game does not give a message when exited without dying and does not have a guide on the snake foods, making the game less playable than otherwise. All in all, these are minor observations, their code works well overall!

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 of C++ OOD approach:

   - Encapsulation: Classes allowed us to bundle data and behaviours making the code less dependent on a scattered range of global variables and much more readable.
   - Easy to expand on: Features like the snake body growth and special foods were easier to extend due to the modularity of the code.
   - Reusable: One class could be reused for multiple purposes. For example, we used the objPosArrayList class for both the snake and the food bucket.
   - More manageable code: Due to distinct separation in classes and their associated behaviours, it is easier to manage individual code blocks, update old code, and debug erroneous code without having to change the entire project.

Cons of C++ OOD approach:

- More complex: With all the classes, member functions, and the interplay between various objects, it is a lot harder on a cognitive level to keep track of all aspects of the project in contrast to the linear design of the procedural code used in PPA 3.
- More dependencies: The interplay between different objects can also introduce dependencies such as the dependence of the player class on the gamemechs class. Such dependencies can hinder modular changes specific to these classes. In contrast, there are few dependencies in the PPA 3 procedural code.
- Overengineering solutions: Even small issues like handling simple game mechanics are a lot more complex with having to create a separate class with its own unique member functions. Whereas procedural code just requires a few global variables in simple functions to emulate similar behaviour.

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.

   Many sections of the code are commented sufficiently with enough information given to the reader, allowing them to understand what each line should do and is meant for. Be that as it may, there are many missing comments for both header and .cpp files that may not require any comments only if people with coding experience see them. However, to people who are unfamiliar with code, there will be much confusion within the code within constructors, and even constructors themselves.
   The best way to improve this issue is to remember to include comments for the overall purpose of each constructor, like writing a very brief description of what each constructor does within objPos.cpp, and explaining complex logic operations, such as the operator= constructor within objPosArrayList.cpp. Additionally, as explained before, the delay constant was defined as 50000 µs, but the loop delay function has a comment mentioning 0.1s, which will cause confusion. Therefore, it would be best to quadruple check the comments in case something does not seem right.

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 great indentation, add sensible white spaces, and even deploys newline formatting for better readability. The code appeals to the reader and looks simple and precise without too much or too little white spaces. Each line of code seems to be indented properly, within its correct spot, and each statement, noticeably, switch cases, are organized beautifully and in a synchronous manner.

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 explained previously, the user-friendly style is not entirely met as no message is given when exiting without dying and the game does not have a guide on the snake foods. However, the snake game offers smooth, bug-free playing experience with no visible issues whatsoever. The snake game functions as it should with each iteration of the project completed properly, in addition to the bonus. However, there is one noticeable, minor issue when playing the game; as the snake eats a food object, its body will grow. However, the time it needs to grow is not instant, but rather takes a tad bit shorter than a second. Due to this small instance of change, only a few would have noticed this potential problem. Although this issue does not seem to affect the game negatively, I believe it would be best to debug this issue by using the debug method and add a breakpoint on the line where it involves the snake touching the food object. Alternatively, slowing down the game by increasing the delay can allow the coders to see the evident issue and debug it.

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.

   Using drmemory, there is no memory leak within the game. Each new operator used to allocate memory have been destroyed from the heap using the destructor and implementing the delete operators from within.

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?

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.