COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members anand4 and dhodiv

Team Members Evaluated yuw79 and bandaa2

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 main program loop in the code is clear and well-organized, making it easy to understand how the different parts of the game work together. The design follows object-oriented principles, with each object handling its own tasks. The layout maintains organization and allows us to get a good understanding about the loop from the ode logic. One thing to improve on could be to comment on some of the objects and explain their reliance on one another.

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.

   C++ OOD Approach

   Pros

   - Using C++ OOD approach allows us to create classes which make code more organized
   - We can create more complex functions and increase program size easily
   - Debugging is easier as you can isolate each object

   Cons

   - Code was more complicated to make
   - More memory leakage issues
   - Usage of classes made everything dependent on each other thus everything had to be implemented correctly for our code to work

   C approach

   Pros

   - Less complexity when creating the program
   - Memory leak issues are less common
   - You can edit the logic for every specific case and tailor it to your specific needs

   Cons

   - There is an emphasis on global variables which make it less organized

- For large data or to increase the size of the program its quite difficult to add new features
- Theres a lot of repetitiveness in the code

Peer Code Review: Code Quality

1. **[3 marks]** Does the code offer sufficient comments, or deploy 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.

There is certainly room for improvement in the documentation of this program. Some functions are well commented such as “food.cpp”, but others like “Player.cpp” are missing comments in some areas. “Project.cpp” doesn’t have comments at all. Overall, we believe that this code is difficult to understand at first glance due to a lack of documentation. We would improve it by ensuring all functions are well documented such that a beginner can go through the files and gain a basic understanding of the flow between files and the overall process. Specifically, we would do this by ensuring that within the functions there is documentation where the important calls and loops are to give the user insight into what that section of code is achieving.

2. **[3 marks]** Does the code follow good indentation, add sensible white spaces, and deploy newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.

Yes, the code follows good indentation, adds sensible white spaces, and deploys newline formatting for better readability. We can easily observe text on separate lines that do not join with each other or the game output.

Peer Code Review: Quick Functional Evaluation

1. **[3 marks]** Does the Snake Game offer a 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)

Yes, the gameplay is smooth and bug-free. Points increment as needed, food generates properly, size increases when food is eaten, and movement is correctly bounded. The game has a working “game over” when a self-collision is detected and the final score remains available.

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.

   Their game does not have any bytes of leak which means they effectively allocated and deallocated their memory.

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?

No, it is not sensible as using Pos makes the code more complicated and adds more lines due to the fact we need to manage the pos pointer. Also, there is the fact that the allocating memory for pos is creating overhead makes the use of Pos struct more unviable.

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(s).</u>

The way we improved the design was by keeping x and y as private members of the objPos class, with a method like setObjPos to handle updates to both coordinates and the symbol. This approach ensures simplicity and efficient access and proves to be a more efficient method than the way we previously handled it.

ObjPos
− xPos: int − yPos: int − objSymbol: char
+ objPos() + objPos(xPos:int, yPos:int, sym:char) + setObjPos(xPos: int, yPos:int, sym:char): void + setSymbol(sym: char): void + getSymbol(): char const + getX(): int const + getY(): int const + isPosEqual(refPos:const objPos*): bool const + getSymbolIfPosEqual(refPos: const objPos*): char const