COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Marc Cosma He Zaiyou

Team Members Evaluated suy Mamamenk

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.
   By examining the main program loop, I noticed that they made 3 pointers to instantiate a player, game and food object on the heap and from there those were used for the corresponding functions. Most of the designs choices i think where good and the corresponding actions where done in the corresponding functions. For example, drawing the screen in a draw screen. The only part that could be negative is that the draw screen function uses a very nested logic which can be hard to follow.
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 pros of the C++ OOD approach is that it could make team collaboration easier because you can split the work in more manageable modules and also for larger projects it could be easier to maintain and you can use those classes for further development if needed. On the other hand, C's procedural design is a lot easier to implement and the logic is straightforward. Thus, developing this in c++ would take longer but as a result it could be better in the long term, while in C it is a lot easier but might not be as good in the long term.

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 supplies sufficient self documenting coding style. I didn't directly notice any shortcomings but I think some stuff could have been written in a simplified manner. For example, in player.cpp instead of checking for if mydir is stop or left or right, you could have just checked if it is not down(!down) which would have looked nicer but both work.

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 and has indented all the functions well. There are sufficient white spaces when needed and overall I see no issue with it.

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)
   For the most part it was good, all the conditions were well added. The only thing I would recommend is to make the board size bigger because with the condition when it is "*", it becomes really hard to control it with a lower board size. Although this could be easily changed by changing the board size x and board size y value in GameMechs.cpp so it is not really an issue more so a suggestion.
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 memory leakage for this snake game. The only potential leak is caused by calling the ncurses library but that has no effect on the code they wrote and this is standard for all programs because all of them call this library.

```
==421== LEAK SUMMARY:
==421==    definitely lost: 0 bytes in 0 blocks
==421==    indirectly lost: 0 bytes in 0 blocks
==421==      possibly lost: 201 bytes in 3 blocks
==421==    still reachable: 62,270 bytes in 509 blocks
==421==                       of which reachable via heuristic:
==421==                         newarray           : 6,416 bytes in 2 blocks
==421==         suppressed: 0 bytes in 0 blocks
==421== Rerun with --leak-check=full to see details of leaked 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?
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).

   This could be done a lot more simply by replacing the pointer Pos* pos with a direct Pos pos as a member variable. This change eliminates the need for dynamic memory allocation on the heap, making the code easier to maintain. Without using the heap, there's no need to implement destructor or handle the Rule of Six (copy and move constructors/assignment operators), which reduces the complexity and minimizes the risk of memory leaks. This makes the design become more straightforward, which is useful for such small and lightweight object.

   UML diagram

   objPos

- Pos pos

- char symbol

+ objPos()

+ objPos(int xPos, int yPos, char sym)

+ setObjPos(int xPos, int yPos, char sym): void

+ isPosEqual(const objPos& refPos): bool

+ getSymbolIfPosEqual(const objPos& refPos): char