COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members: Adeel Ahmad 400517284(ahmaa20), Shiv Mahida 400522442 (mahidas)

Team Members Evaluated: Aryana Akhlagh 400511949 (akhlaa4), Dora Mackie 400497449 (mackid5)

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.
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.

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.
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.

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)
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.

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. <u>You are expected to facilitate the discussion with UML diagram(s).</u>

Peer Code Review: OOD Quality

1. Code is extremely well organized and easy to follow. There is nothing too unusual about it except for the fact that the destructors are called for the pointers rather than a delete function, which works and makes sense however is a little unusual. However, since it works, there is no further complaints from us.

2. Pros would be that it saves storage as you can store a variable inside a class rather than making global/local variables, it saves lines of code because you can just call a class instead of doing multiple complex functions, it makes the code more efficient, debugging functions is more time efficient and the complexity can be increased in a more meaningful way. Some cons to this approach would be that it is sometimes a little harder to read as calling an object, especially with multiple pointer elements, can be a little confusing for people unfamiliar with the code. Also, when debugging or tracking code, jumping back and forth in certain situations between multiple classes with many functions, it can be a little confusing at times. Other than that, OOD approach seems to be a lot more organized and efficient of a manner to code in.

Peer Code Review: Code Quality

1. Commenting is enough for me to understand and not too bloated for it to be excess, Code could be a little more descriptive, especially in the player function, and the commented-out functions/failed code should've been removed in advance to improve the organization and readability of the code.

2. Code is excellently indented, spaced out, and has good white spaces for perfect aesthetic and readability in all of the classes as well as the main function. Nothing we would change here.

Peer Code Review: Quick Functional Evaluation

1. When playing the game, everything, from the collisions to the wraparound to the score increments to the special items to the snake elongation, everything went smoothly. The only problem, which is hard to recreate, was that sometimes it was possible to go in the opposite direction (Ex. Right to left), which sometimes caused a normal 180 and sometimes caused a crash into the own snake body (an automatic, random death which wasn't due to negative score). There was also some instances where multiple inputs in a row were not accounted for, and an instant death was caused. When looking at the code, the direction function as well as the get input is not cleared IMMEDIATELY after an input is taken, leaving some ample time for the code to accept multiple inputs, which leads to this bug being very persistent, especially later in the game. One way to fix this issue is to try to clear the input immediately after the direction is processed or clearing it multiple times in the code to ensure that there is no room for multiple inputs to be selected.

2. No leaks, amazing!

Project Reflection

1. Yes, we believe that the current compound object design of objPos is not as sensible as it could be as in our opinion it would be just as sensible to use x and y as regular public field members, decreasing complexity and memory allocation used. This is because in c++, a struct is just a very low-level instance of a class which creates its own object which would make sense if it was a much bigger, more complex data type which would be in need of organization. However, since there is only position elements x and y, along with pointers to the struct, it creates more unnecessary complexity to the code. There would be no sacrifice to organization, implementation, or readability when converting these to public field integers as well as ease debugging to a certain extent. It can be said that there would be less ease of dynamic memory allocation, however with the help of objPosArrayList and intuitive, easy implementation, there will be no sacrifice to space allocated on the heap.

2. The way I would improve this design is by removing the Pos object all together, as well as implementing the positions x and y as public field members.

objPos
+ x: **int**
+ y: **int**
+symbol: **char**
+ objPos()
+ objPos(xPos: **int**, yPos: **int**, sym: **char**)
+ objPos(o: **objPos**)
+ operator(o: **const** objPos): **objPos**
+ ~objPos()
+ setObjPos(o: **objPos**): **void**
+ setObjPos(xPos: **int**, yPos: **int**, sym: **char**): **void**
+ getObjPos(): objPos **const**
+ getSymbol(): **char const**
+ getSymbolIfPosEqual(refPos: **const objPos**): **char const**
+ isPosEqual(refPos: **const objPos**): **bool**

This design, along with the objPosArrayList, which creates dynamically allocated arrays with intuitive relations to the snake position, will allow for a much more smoothly flowing, more innate way to code the position function within the project.