COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Aegir Wang Joshua Xue

Team Members Evaluated Akinniyi Chidiebube

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.

Yes, the main logic is very clear and concise. One part of the code that was a bit long was the "DrawScreen" function, where each character needs to be checked for either being a snake, a food, or a blank space. Otherwise, the code has little unnecessary parts and is easy to follow. They also followed the rule of minimum 4, implementing a default constructor, copy constructor, copy assignment operator and destructor.

Another negative thing we noticed was that the person only made a total of 1 commit to their GitHub repository. This project was meant to exercise **incremental engineering**, meaning it would have been better to make a commit every time a new function or iteration was finished to “save” the code. However, the code itself had no issues, and was clear and concise.

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++:

- Inheritance: Gives the ability to reuse code
- Encapsulation: Easier to read/better code organization
- Easier to narrow down errors/easy to maintain due to object orientation

Cons of C++:

- More complex, not a good choice for smaller projects
- Needs to follow the rule of minimum 4 for classes

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 has sufficient comments. However, the code assumes that you have a good understanding of how the logic works already, as explanations of the logic are not written. Only a general idea of what the code does is written. This would make the code very easy to understand for a more experienced programmer but leaves a lot to be desired for a beginner programmer.
There's no proper way to fix this other than to know who you are writing the comments for. The complexity of your comments would entirely depend on the skill level of the person reading 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.

Yes the code is actually formatted very well. Anything that needs indents is there, and each function is spaced properly from the surrounding functions. Curly brackets {} for each function also all follow the same trend, all written on their own separate line. The organization of the code makes it very easy to find functions and classes.

One shortcoming is that they did not delete the default comments that were always there. Now that the code is done those comments can be removed. However, overall, everything looks pretty good.

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 Snake Game offers a smooth, bug-free playing experience. After playing the game multiple times, we did not detect any buggy features.

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.

When running DrMemory, there were 0 bytes of leakage found. The code is written well, and anything allocated using **new** was deallocated using **delete** or **delete[]**. 2D arrays were also properly deallocated using for loops.

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?

I think that the objPos class did not need an additional struct to store the Y and X values, and could have just been implemented as members of the class instead. Using struct Pos induces dynamic memory allocation pointing to the struct, which increases heap memory usage. Not to mention the risks of memory leaks if not deallocated correctly.
It is redundant at best and dangerous at worst. As electrical and computer engineers, our code should be "low level" and close to the hardware, where abstractions such as object oriented programming may cause unnecessary overhead.

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).

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

This alternative design to the objPos class follows better encapsulation practices. by putting x, y and symbol in private, we can prevent accidental modifications to the snake's position and symbol. By doing this, we remove the need for the struct that has to be called in the objPos class.

A getter to get the x and y positions are also added at the end. This makes the code function the same as before, but is much simpler without the uses of the struct. Because of this, the destructor can also be empty.