COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Lucy Schartner (schartnl), Erin Herzstein (herzstee)

Team Members Evaluated: narends, melnbaa

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.

<u>Peer Code Review: OOD Quality</u>

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.

It is very easy to interpret how the objects interact with each other in the program logic through the code. For instance, GameMechs and Player objects interact with each other in the initialization routine to generate food. It is also evident that the OOD quality is high, and the code is never decomposed into a procedural design approach rather than an OOD approach.

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	C Procedural Design Approach
Pros	● Often easier to maintain, ● Easy to expand from (e.g. ability to create a food class to expand on in the bonus) ● Encapsulation allows for flexibility in access and error-prevention (private members accessible within the class, public accessible from outside) ● More succinct code due to	● Simple to implement, especially for smaller programs (i.e. computing a number from a set of variables) ● Often lower memory usage and faster execution ● More accessible to beginner programmers

	inheritance and polymorphism (can reuse code)	
Cons	● May present unnecessary complexity; for a simple program (e.g. calculating a number through a formula), having global variable definition may be easier ● Less intuitive programming, requires a more enriched understanding of programming ● More complex object relationships mean that the programmer has to be very organized to prevent memory leakage	● Less structured code (no encapsulation) makes it more difficult to collaborate with others and to maintain the program ● Poor scalability; difficult to add on new features without changing the full structure

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.

The names of variables and objects are very descriptive so commenting is not necessarily required to explain the code. However, some more comments may have improved the readability of the code, as there are very few comments included. For example, some comments to explain the nested for loops in the DrawScreen logic may be helpful.

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 good indentation, adds sensible white spaces, and deploys newline formatting for better readability. No shortcomes are observed.

<u>Peer Code Review: Quick Functional Evaluation</u>

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)

The Snake Game offers a smooth, bug-free playing experience. The wrap-around is seamless and there are no awkward pauses or delays while playing the game.

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.

The Snake Game does not cause any memory leak.

<u>Project Reflection</u>

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?

In our discussion, we decided that the compound object design of objPos is not sensible, as it adds unnecessarily complexity without good reason. The addition of a pointer dedicated specifically to the position could be a memory-conserving strategy; however, in the context of this project, we always instantiated a position for our objects. Having the additional Pos struct required added effort, as we would refer to the Pos struct first instead of directly accessing the position (which is done for the symbol). Thus, we found that including the additional Pos struct was unnecessary for this project, as it introduced more work for the developer without a significant benefit.

2. [4 marks] If yes, discuss 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 a UML diagram.</u>

To improve the object design, we think it would be best for the objPos class to have separate variables for x, y, and symbol. The following UML diagram helps explain our design, and our suggested changes are in red:

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