COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Adam Jasinski, Akashdeep Singh

Team Members Evaluated Yusuf Rashwan, Yousif Fadhel

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

Part I: OOD Quality

1. **[6 marks]** OOD is about sensible code modularization. Looking at the header files of each object, can you easily interpret the possible behaviours of the objects involved in the program, and how they would interact with each other in the program? Comment on what you have observed, both positive and negative features.

   Most of this group's header files for each object provide comments on why a library has been included and a brief explanation of what a block of member functions does, and if no comments were provided, it was easy to understand what the object may do due to the good naming conventions used. One drawback we see in the header files is that this group used the #define command in some of their header files, which is a preprocessor, to create macros. While this is not inherently incorrect, it can be more challenging to debug and is prone to errors or can lead to unexpected behaviours. A better alternative would be using the const qualifier which provides type safety and it is easier to debug. Additionally, The values of the board size are hard-coded, and the srand() function, which is used to seed random number generation, is used inside the random food generation function, which when called repeatedly can cause the same number generation. Again, this is not incorrect, although it is good practice to call it only once in the constructor or the main function. Other than that, it's very easy to identify the possible behaviours of the objects involved and thanks to the many comments and good naming convention it's easy to understand how they interact with each other.

2. **[6 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 is well structured. It follows a modular design thanks to the use of classes and it's well-commented making it easier to understand. All of this contributes to making it easier to understand how and why each object is used and how it interacts with other objects. Additionally, the names for each of the objects make it clear what they are referring to. The objects are efficiently maintained, they're initialized before the main logic part and properly taken care of in the clean-up function. The only negative feedback we noticed is on line 72 of the project.cpp file where there's the use of "magic numbers" that is, values that are not defined anywhere and just appear in the program; these can be replaced using enum.

3. **[5 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.

   Procedural programming is simpler and uses more straightforward logic, and can therefore be easier to understand, that is until the program gets too complex. This simplicity leads to a few shortcomings, including heavily relying on global variables, and limited reusability due to the absence of objects and classes, which leads to code repetition in multiple parts of the project. This can be avoided using an OOD approach as seen in the project. Defining different classes as this group did in their code allowed them to add a level of abstraction, which means hiding the dirty work in classes and creating an easier-to-read and follow main program loop. The objects defined in one class were used in other classes, making the code reusable. Abstraction and code reusability thanks to objects lead to a more modular and better-structured code. Overall, using an OOD approach led the group to have a simpler and easier-to-follow main program loop with the complex and dirty work "hidden" in objects and classes, therefore highlighting only the important features of the program.

Part II: Code Quality

1. **[5 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.

   After reviewing this group's code it is clear they have put a lot of effort into making it self-documenting and also added comments. All of their variable and function names are made so that you know what it is referring to by just reading the name, and they refrain from using random values or numbers that just appear out of thin air. Their code is very well formatted and follows a standard structure throughout, making it easier to follow. The only drawback we can see is that some of their bigger and more logic-heavy functions, such as their generate food function in the food.cpp file, there is a slight lack of comments and it would be nice if they had added just a couple more explaining the logic. They have general comments on what the function does, but sometimes it can be hard to follow how their logic is working and requires extra observation to fully understand. Overall the code is very well structured and follows a good self-documenting coding style, however, some additional comments in the more logically heavy or longer functions further explaining what the code does would boost the overall clarity of the group's code even further.

2. **[4 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.

   This group's code does follow good indentation, adds sensible white spaces, and deploys newlines where needed. Through in-depth observation of the code, it is very easy to read and

analyze because this group uses indents whenever there are any loops, conditional statements, etc. and they also make sure to add new lines whenever a new larger code block is about to begin, as to not clump it together with another block. They also use white space correctly, as they space out separate functions and logic in their code, to differentiate one part of the code from another. All of this together creates an easily readable code that almost anyone can follow along with quite easily.

Part III: Quick Functional Evaluation

1. **[8 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 a technical user feedback)

   By testing the functionality of the game multiple times and trying to find any bugs, we have concluded that this group's snake game works very well. There are no program crashes, or logic errors in the snake movement, collision, or growth, and the game runs very smoothly and is not too slow or too fast. The exit conditions are satisfied and there are no errors that occur with them as well. Additionally, this group has completed the bonus and it also works very well. The bonus looks to have 2 special foods, with 1 increasing the snake length by 10 and adding +5 to the score, and the other food simply adding 10 to the score. Overall, this group created a very enjoyable, smooth, and bug-free experience for the user.

2. **[6 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause of the memory leakage.

   By using DrMemory we can see that the group's project has no memory leaks. Their destructors are properly coded and do not delete anything more than is needed off of the heap. Additionally, they use proper C++ syntax for their destructors.

Part IV: Your Own Collaboration Experience (Ungraded)

1. Tell us about your experience in your first collaborated software development through this project – what was working and what wasn't. If you are a one-person team, tell us what you think may work better if you had a second collaborator working with you.

   As this was our first time working on a collaborated software development project, it was quite a good experience overall. We enjoyed the parallel coding experience as it allowed us to work on our own time, while not needing to rely on the other person's code entirely. We both think the biggest thing that wasn't working was the fact that later on in the project, when we needed to make the snake body, collision detection, etc. since we did not directly code the entire project, there were gaps in our overall understanding of each others code, as obviously, we did not write it ourselves. This led to some issues while writing new code as we would sometimes use

functions incorrectly due to our lack of understanding. However, overall, we both enjoyed this experience and we were able to create a functioning snake game that both of us are proud of making.