COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Humza Noor Mustafa Mazhar

Team Members Evaluated Eloise Nguyen Kiana Wilding

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.

   ANS:

   Looking at the header allows us to get an in depth understanding of each object as each method has a name that accurately depicts its functionality. Furthermore, the header files clearly show the relations between different objects, has a and owns a relationship. Header files follow the manual and consequently allow the reader to easily grasp what the object does. Methods in the header files are organised in groups by coagulating getters and setters. In GameMechs.hpp they define a destructor however they leave unnecessary comments in GameMechs.cpp and the destructor doesn't actually do anything which is a pretty unconventional practice in C++. Normally if you want to use the default destructor you don't need to declare as it is already provided and if you would like to declare the destructor to improve readability it should be declared using = default, which would mean that no implementation is required in the .cpp file.

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.

   ANS:

   The main loop seems to make sense as the logic follows the instructions given to use in the manual. However, the code seems to be very disorganized as there are a bunch of unnecessary debugging comments that have been left in the final code. These debugging comments should have been removed before the code was submitted as it may confuse readers who have not read the project manual. Furthermore, they should add comments in the main loop to make it easier for readers to understand the code. Moreover, there seems to be a random number of new lines after each few lines, normally there is one new line or a fixed set number of new lines.

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.

   ANS:

Pros:

- Encapsulates data and functions into objects, promoting modular design.
- Allows for code reuse through inheritance and polymorphism.
- Easier to maintain and update due to encapsulation and abstraction.
- Better suited for large, complex systems where relationships and behaviors are important.
- More naturally models real-world scenarios with classes and objects.

Cons:

- Can be more complex to understand and implement, especially for beginners.
- May have performance overhead due to features like polymorphism and dynamic binding.
- Typically consumes more memory because of object overhead.
- Provides less control over hardware and low-level functions compared to C.

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.

   ### ANS:

   While the code does offer comments they tend to be inconsistent across all the files. The comments that were written tend to be brief and are not very beneficial in understanding logic. For next steps, we would urge that detailed comments are used to explain non-obvious logic, inputs and outputs etc. Lastly, we feel that large amounts of commented out code(which may have been used during the code development for debugging) were prevalent which could have been removed to give the program a cleaner look and removing any possible confusion.

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.

   ### ANS:

   The code does generally follow good indentation making it easy to distinguish between code blocks. With reference to whitespace usage while it is used, consistency of use is very important hence for the next steps I would recommend an increase in the usage of white spaces to separate logical codeblocks. Lastly, for the vast majority of the code there are a random number of newlines placed in between function calls which is unnecessary. Furthermore, there are a random number of new lines after each function which is not a common practice in developing

projects.

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)

The game offers a smooth experience, correctly implementing the draw screen, wrap around, increase in length, random food generation and suicide logics. The escape key is set up to force exit the game. The movement of the snake is responsive to direction inputs.

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.

There was no significant memory leak on running a memory report.

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.

ANS:

Our experience as a team was very smooth and without any major hiccups. Consistent communication and keeping each other updated on the others' progress worked well, as well as working together on trickier parts of the code. Using the method in the manual, dividing the workload in the earlier stages into two parts and completing individual tasks worked well and allowed for a streamlined process of code development. Time management was the only issue, as the calculus midterm took up a lot of our time which would have been otherwise sufficient to complete the code. This could have been prevented by allotting less time to the earlier or individual pieces of the code and more time to the final stages of the code and debugging.