COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Yax Patel, Sharvin Soosiapillia
Team Members Evaluated	Megan Saunders, Kendall Bird

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

Part I: OOD Quality

1. **[5 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.

The head code overall is very easy to read and it is easy to understand what the class member functions are there for. Something that could be changed are the older comments from the boilerplate. When reading the header files, these would not provide any value to an external reader.

2. **[5 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.

In the main function loop, the logic seems very straightforward. All of the different functions seem to be well organized and follow a very straightforward self documenting code style. For example, in the DrawScreen function, it is very easy to see that at the beginning, some temporary variables are created for food and snake position to be used later on. Then the whole board is looped over to draw each of the cells with the appropriate contents. Overall very minimal comments are required. In terms of program logic interpretation, I cannot find any negative feedback. Everything seems very easy to read.

3. **[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:

- less code rewriting (reused pieces of code)
- cleaner code overall
- safer (getter and setter methods)
- modular code, very easy to reuse (ex. create a new player)

Cons:

- can take longer to write code for very small programs
- code can get messy if not well thought out beforehand

Part II: Code Quality

1. **[4 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.

When analyzing the code provided, there are very sufficient comments that are provided. as they are able to summarize what a section of code is performing with a single single comment. Additionally with more complex lines of code (i.e multiple things happening on a single line), they provide a comment of that particular line to allow the developer to get a key understanding of their thought process and their solution. An example of this would be in the player.cpp file, where they provide details about attributes about their code such as their adding/removing the tail and checking to see if the player has collided with themselves.

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.

Not only does the code have more than enough comments to explain the process of work, but it is also very well formatted to allow for easy readability. The only suggestion that should be implemented is to include a bit more whitespaces within their program, this allows other developers to understand that they are moving on to a different piece of code (such as a different function or a different conditional statement). This is primarily evident in your project.cpp file To improve this, simply add a bit more spacing between key components of code to allow for easier readability.

Part III: Quick Functional Evaluation

1. **[6 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 a technical user feedback)

From a technical point of view, the snake game runs pretty smoothly as the frame rate chosen allows the game to be very smooth. However, from a gameplay functional point of view, there are a couple of small errors that impact the project. To begin, it can be seen that the snake can go into the border, which is not expected behaviour. Another behaviour that was noticed was that the food always spawns in the same order locations every time you replay the game. This is due to the same seed being used every time by the rand() function. This could be fixed by adding `srand(time(NULL))` to the initialization function. Another problem that was faced was that when the player lost the game (ran into itself), there was no message showing the user their final score. This would be better since you don't really take a look at your score while playing the game.

2. **[4 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.

In terms of memory leaks, by running dr.memory there appears to be a memory leak however this is not a leak from the game itself, but rather it is from the compiler attempting to allocate memory. This is not a major memory leak as it is an unavoidable issue. Otherwise, there are no major memory leaks.

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.

The collaborative experience using git and GitHub was great. It allowed us to work on different code features in different branches and merge them to the main branch once we were done. It also allowed 2 individuals to work independent of one another and ultimately come together at the very end to create a final product more efficiently.