COMPENG 2SH4 Project – Peer Evaluation

Your Team Members _Matthew El Chalouhi___ __Tyler Smith_____

Team Members Evaluated ____Justin_____ ___TJ_____

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.

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.

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.

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.

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.

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)

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.

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.

Part 1:

Generally, their OOD implementation seems to be simple, effective, and easy to understand. Reading all of the header files, the functions have a clear purpose and interact with one another cleanly. I like their use of adding a separate header file for all global definitions, this makes it so that they are more easily accessible and all contained within the same place if they choose to edit them later. However, these definitions were not used in any of the class header files and was only utilized in the project.cpp file, and certain global definitions were still made in other header files, so overall a good idea but poor execution.

In the main program loop, there are some negative features however, they use the numbers 32 and 126 rather than the ascii characters they represent, this makes the code much more difficult to understand especially if a person does not have an ascii table readily available while debugging. Additionally the message that is printed at the bottom is done so using a lot of concatenation of strings that is difficult to read and does not follow common programming structure.

The OOD approach of the project has the benefit of improved encapsulation and code organization and readability. It makes it much clearer what is going on at any given moment and prevents different variables from accidentally interacting with one another or editing one another as with structs. It also allows for more powerful built in methods to be included in the class definition rather than in the main cpp file, which remains reserved to the more core logic and functionality of the program.

Part 2:

The code by this team had generally very good readability, and almost every instance where it wasn't instantly intuitive what a certain section of code did, there was an accompanying comment to explain it. The whole program was very simple to read and understand, with elaborative but concise comments, and descriptive function and class names.

However, in the Player.cpp file, the self collision logic is not contained within its own method, but rather is placed within the move player function. The implementation for the self collision is very unlike what we did in our function. Justin and TJ implemented their collision check by checking if the next coordinate

the snake will move to has the same character as the snake. This is very good in terms of efficiency, because it is only checking 1 single coordinate value, but their implementation results in a lot of repeated code for each of the 4 direction switch cases, resulting in code that is difficult to understand and potentially debug. Additionally the 'total' variable in the project.cpp file is just a large block of concatenated strings that can be difficult to understand and breaks away from typical code formatting in the rest of the program, it would have been better if they looked for alternative ways to print this message, or just kept the message in 1 line rather than having an odd looking block in the middle of the project.cpp file

Other than these issues, I could not catch any other problematic sections of code. The rest of the project has excellent use of indentation and whitespaces, with very good readability, in addition to having good commenting and follows a self-documenting coding style where the function names are self descriptive.

Part 3:

After playing their snake game about 50 times, testing the wraparound, collision, and food collection; I could not find any noticeable bugs in the program. The wraparound logic functions correctly, the detection logic functions right away when there is a collision without printing any extra icons to the gameboard, and the food collection and growth works flawlessly. Overall the game runs very smoothly, likely due to efficiency gains in the collision logic as I specified in section 2, with much less jitter and strobing than many other implementations I observed.

However one thing that must be discussed is that the snake game still contains debugging messages that are printed out, this leads to an inferior playing experience as there is a lot of unnecessary text at the bottom of the screen.

Every time new allocations on the heap were made they were properly destroyed, this was confirmed by running the project.exe in drmemory where it observed 0 bytes of memory leak.

Part 4:

I think that the project could have easily been an individual assignment, it is too small to properly learn how to coordinate with other people when working on git repositories and by working with another person there is less learning taking place of OOD within C++ since the amount of code you are writing as a single person is less; this is especially felt since this was the first time we are writing C++ code without the guidance of the lecture, and so by the time you finally begin to feel comfortable with the code you have nothing left to work on. We personally found the advanced functionality a bit too tricky to work on because each of us didn't fully understand how the other person's code worked, I think that had we worked on our own games independently, it may have taken longer, but not by a significant amount and the learning would have been improved as the concepts would have been reinforced further.