**COMPENG 2SH4 Project – Peer Evaluation**

Your Team Members \_\_\_\_\_\_\_Alina Li\_\_\_\_\_\_\_ \_\_\_\_Hanna Nopper\_\_\_\_

Team Members Evaluated \_\_\_Maryam Ismail\_\_\_\_\_ \_\_\_\_\_Isha Patel\_\_\_\_\_\_

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.

There are five distinct header files (excluding the provided “MacUILib.h” file). The developers followed the second design choice using “Food.h” then referencing it later in “Player.h” (as described in Iteration 2B of the Project Manual). By investigating both public and private classes, it can be seen that the appropriate “objPos.h” and “objPosArrayList.h” are both included in the overall project execution. Overall, the functions and variables are clearly labeled and easy to follow, making following how each element interacts relatively simple and straightforward.

However, due to the lack of comments in the header files, an unexperienced programmer or one who does not have an understanding of this specific project might struggle analysing the code. This could cause further issues should a third developer start work on the project as it could take significantly longer time to interpret. Adding comments to explain what each second does, or any abnormal fragments, will simplify any future analysis.

1. **[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.

The code in the main loop is modularly structured, which demonstrates good programming practice. By investigating the header files, we can see how “Player.h” incorporates the remaining files, which demonstrates a strong understanding of separating the code into modular fragments. The main loop is structured to get the starting input, run logic, draw game screen, reference GameMechs.h and Food.h to operate game, then cleanup when the game quits. As discussed below in Part III, there are some elements missing in the main game program which hinder the game play – specifically the exit key to finish the game.

1. **[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:

* C++ OOD allows for the creation of classes to be kept separate, allowing the developers to work on multiple classes and steps without affecting the program as a whole.
* The class setup helps reduce code redundancy, while being able to reference each class multiple times.
* Inheritance allows the classes to follow a hierarchical system, prioritizing some logic over others.

Cons:

* It was challenging to distinguish between C and C++ at first which increased the time to code initially.
* Larger memory due to classes which slows down the program
* Due to the abstract nature of C++ OOD, it took some time to wrap our heads around the high level understanding of the flow

**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.

In general, there are minimal comments which made following the code at first more challenging. As mentioned above, the lack of comments could provide issues for further development. However, since there are clear names used for variables in functions, with some effort, the code can be easily deciphered.

1. **[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.

Yes, the code follows all these mentioned readability techniques. While the lack of comments (as mentioned before) makes the code slightly harder to follow, it is more readable. Since there are no shortcomings, there are no further recommendations.

**Part III: Quick Functional Evaluation**

1. **[6 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)

When attempting to compile the program, we encountered an error (see Figure 1). Through further investigation into the makefile, we found that they are missing “food.o” which is why the program couldn’t compile. The developers need to add this back in to be able to compile the program. It should be noted that the terminal output stated that there is a C input being used in the C++ program. This is not something we further investigated because once we fixed the makefile ourselves, the program was able to run.

A screen shot of a computer program

Description automatically generated

Figure

The game itself ran relatively smoothly after some delay. The delay is most likely caused by the significantly larger data processing which we have not covered within CompEng 2SH4. Their instructions to play the game were clear and concise, matching the theme of the game. Operating the snake was simple and it moved at an appropriate rate. We tested the snake colliding with itself which caused the game to end, as expected, but there was no notice or slogan other than the game simply quitting.

There were a few elements of the game missing. There was no “win” or “lose” slogans, which is necessary for the game’s purpose. The game did keep track of the score well but, other than colliding with yourself, there is no way to end the game (no exit key). This should be programmed into GameMechs.h . Overall, the game functioned relatively well without any noticeable bugs, when it did run, but is missing a few key elements.

1. **[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.

There are no identifiable memory leaks in the program, but there is a significant time delay which could point towards leakage somewhere in the code.

**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 it was both of our first times working collaboratively, we had to work on our communication to figure out who does what. In the end, we figured out that sitting together and working through the code at the same time was what worked best for us in this circumstance. However, we are both aware that this is not the most efficient way of working through a project. Ideally, we would have divided up the work so each of us works on individual components (parallel software development). For example, once the header file is developed and we agree on specific variable names, one person could work on the snake’s movement while the other works through the point system. As suggested in the project manual, we attempted to follow the recommended workload breakdown for Phase 1 (Dev.1 sets up game mechanics while Dev.2 implements player objects) but we found that discussing the individual code elements while writing helped us both fully understand what was happening. Moving forward, both of us want to work on being more independent and trusting our knowledge and skills.