COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Sankavi Kirushnakumar and Caylia Bonnick

Team Members Evaluated Fatima and Sophia

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.

By looking at the header files of each object, it is easy to interpret the expected behaviours of the objects as well as their interactions. They are laid out clearly and include the needed classes when necessary, making it apparent how each file will interact with each other. In each header file, the functions are given clear names that make it easy to identify the action each Is supposed to eventually execute. However, there is a bit of confusion to note. Specifically, in the "Player.h" file, the implementation of this file repeats itself. This makes it a bit confusing to read and interpret as it seems like "Player.h" was defined twice. This confusion could have been avoided by deleting the repeat or at least commenting it out.

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.

When examining the main program loop logic, it is easy to follow along with the logic to interpret the way each object interacts with each other and how they all work together to make the program run. The name of each function helps with easily being able to identify what should be going on as each line of code is read. An issue, however, is the fact that the program lacks comments. In sections of the program that get more complex such as "RunLogic" or "DrawScreen" it would be helpful to have comments, even just a few, to give the reader an easier time following along with what is happening.

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	**CONS**
C++ OOD Approach	- **Modularity**: Encapsulation makes it possible to create classes, facilitating the management and modification of the game's various components - **Inheritance** and **polymorphism** facilitate code reuse, allowing for the creation of more flexible and extensible code. - Helps in modeling real-world entities (like the snake, food, game board) using classes	- OOD concepts has a steeper learning curve - Multiple classes and their interactions can cause the codebase to become excessively complex if it is not designed properly
C Procedural Design Approach	- Quicker and easier to implement without the various classes as seen in PPA3 - The overall familiarity and simplicity with C procedural design was better to grasp and implement	- There is limited scalability as the projects grows further and requires more intense features - When comparing the Project and PPA3, there is a lack of code reusing in PPA3 - The lack of encapsulation causes some debugging issues as dealt with in PPA3

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.

Within the code there contains very few comments, mostly just for straightforward operations. There is overall a lack of comments explaining the more high-level code and rationality behind certain code decisions and algorithms, especially in the main Project.cpp file. Some parts are hard to follow along with where no comments are placed, for example, within the 'RunLogic' and 'DrawScreen' functions where more algorithms and action of the physical program takes place. To improve this, adding more precise comments within each function and specifying the purpose and any important parameters, or the overall action of certain lines of code. If there are any non-obvious operations or intricate logic, explain them in functions using inline comments. By including more comments, it will enhance the code readability and comprehensibility.

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.

Throughout the code there is consistent indentation allowing for readability and understanding and the functions. There also includes newlines for the print statements that shows up when the game is played. Though, despite the good indentation, some function contains lengthy blocks of codes such as the 'DrawScreen' function would could be sectioned/breakdown into smaller segments for better readability and overall understanding of the function. Adding a commented header line above the section for the draw screen where the broader is being draw and the food can be done to organize more sufficiently (e.g, // Drawing Border or Drawing Random Food).

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)

- **Buggy Feature:** Game does not display end game message or final score and manually terminate with end key
    - **Root Cause:** There is potentially a missing or improper trigger end game condition implemented or display logic within the main Project.cpp file
    - **Debugging Approach:** Review you end game conditions and where is it placed, where is the input of the key coming from? Is it ending the entire main game loop since it is a forced terminate key? Hint: Review your exit flag should it be set to true or false, and review getInput()

- **Buggy Feature:** At certain moments in the game, when collecting food the game end message briefly appears for 0.5 seconds
    - **Root Cause:** The collision detection might not be handling food collection accurately, could be triggering some sort of end game condition
    - **Debugging Approach:** Analyze the collision detection and food collection logic. Determine that collision checks are carried out accurately and that game-ending messages are only shown when intended.

- **Buggy Feature:** Incorrect implementation of self-collision, when collision happens the game does not terminate fully and just continues the game
    - **Root Cause:** Incorrect implementation of self-collision detection and game terminate logic
    - **Debugging Approach:** Review the 'RunLogic' function within the main program loop, and examine any differences that may exist between the methods used for game termination and collision detection. Hint: If a collision happens, what do we set the exit flag to?

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.

As seeing the debug report below there are 0 bytes of leaks, therefore the game has no cause of memory leaks.

```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       4 unique,     5 total uninitialized access(es)
~~Dr.M~~       1 unique,    58 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total GDI usage error(s)
~~Dr.M~~       0 unique,     0 total handle leak(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
```

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.

Our experience in this collaborative software development was pretty decent. There were definitely times in which we needed to depend on each other iteration implementation to progress with out our. And the consistent git pulling and pushing with different operating systems (Mac and Windows) could get confusing each time trying to change the MacuLib and makefiles. Otherwise, it was a great experience learning how to collaborate with other individuals on the same code as this is something that is done in projects in the real world in many companies.