COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Yuxiang Chen and Nirmaan Patel

Team Members Evaluated: Nuha and Shanza

Part I: OOD Quality

1. 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 header files for each object look neat and nicely spaced**
- **They have provided and implemented the useful functions for each object, and the variable definition is also done neatly**

However,

- **it would be very useful if a short description of any additional functions or variables is provided beside the function or variable, so that the reader can be clear of any misunderstanding while viewing the function definitions in the ".cpp" file.**

2. 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 logic of the main program loop, the definition of object instances usage of functions of different classes is done very properly, according to needs.**
- **The main logic code looks very short and concise because of the efficient use of classes and objects**

3. 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:

- **Neat & legible code reading and understanding**
- **Don't need to scroll through a large amount of code to find errors. We can access particular files related to the error, and correct them**
- **In ODD, the existed class can be reused repetitively, and even be modified or upgraded by creating subclass derived from it, in other words, inheritance.**

- **Maintainability. For example, if an algorithm error is later found in a member function, all I need to do is to fix the function itself, and the correction will propagate simultaneously to all other parts of the codes that used this function.**
- **Teamwork friendly. Every team member can work on only their only part without interference or distraction with each other.**

Cons:

- **Need to ensure proper linkage between code documents, so that there is proper functioning of the entire code or project**
- **Need to ensure of deallocation to avoid memory leakage, across a large set of documents**
- **Variable naming convention and usage needs to be proper to ensure code readability**
- **Need to provide scope resolution for every function definition of a class**
- **In some cases, for example in a simple hardware level of program, ODD can be an overkill compared to procedural programming.**
- **Programmers must be very familiar with the properties/behavior of each class and how they interact with other class, can be especially challenging for beginners who just switched from procedural programming.**

Part II: Code Quality

1. 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.

- **Adequate description and explanation have been provided wherever a reader might find it difficult to interpret the code**
- **Also, there is minimal usage of extra variables to achieve a purpose, which makes the code concise and enhances readability**

However,

- **at some places, there are still traces of comments that might have been used during the debugging or developing process**
- **these comments seem to distract the reader a bit from the main content of the code**

2. 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.

In terms of the overall presentation of the code

- **the spacing, indentation, and newline formatting is very commendable. It captures the reader's attention to detail**

- **The overall flow of the code, with the help of the comments, is very meticulous, and it feels like reading a step-by-step process**
- **in the header files, the functions have been separated into groups according to similar functionalities, which makes it easier to comprehend**

Part III: Quick Functional Evaluation

1. 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 functioning of the game and the user-experience is very smooth**
- **There are no glitches or disruptions in the game, which suggests that there are no buggy features.**
- **In addition, we noticed that they left some unused variables, for example, "int **myFood" in food class, that were presumably intended for building the bonus part of Snake game. Since such variables did not sabotage the basic version Snake game's performance, it can be considered as "leave the hooks" for potential future feature expansions.**

2. 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.

```
void CleanUp(void)
{
    //MacUILib_clearScreen();  //uncomment if we want board deleted at end

    MacUILib_uninit();

    //remove heap instances
    delete myGM;
    delete myPlayer;
    delete myFood;
}
```

```
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~      20 unique,   335 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 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:
~~Dr.M~~      20 potential error(s) (suspected false positives)
~~Dr.M~~          (details: E:\DrMemory-Windows-2.5.0\DrMemory-Windows-2.5.0\drmemory\logs\DrMemory-Project.exe.27620.000\potential_errors.txt)
~~Dr.M~~      31 unique,    31 total,   7392 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
```

It is known that memory leakage can be caused by unreturned resource occupied by allocation on heap even after the program ends. Thus, it is important to use destructors for the class that creates variable on heap, and to deallocate any variable on heap from the Project.cpp.

We inspected all the destructors & deallocation process and ran a Dr.Memory test. There is no memory leakage detected.

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.

- **Throughout the project, the teamwork and collaboration of the team was excellent.**
- **It was nice to learn from each other while developing the code for the project.**
- **Also, the parallel development of code, and its integration into the main program, was a truly great experience.**
- **The end result (perfect functioning of the game) was the icing on the cake, and a very satisfying moment.**