COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Junseo Mun and Shajeeven Nadarajah

Team Members Evaluated Ryan and Angelo

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.

After looking through the header files, overall, the general organization and interpretability of the code is very simple and easy to follow. The possible behaviours of the objects in the program and how they would interact with each other in the program is clear and neat in terms of code order and location. For example, the team's constructor and destructor member functions are done as outlined by the manual in an appropriate fashion along with appropriate variable names given to functions and data members. One small component that can be improved on is the implementation of the playerLength in the main program, as they hard coded each change in length. This is seen where the change in length is implemented through a for loop in the Player.cpp file. This can be improved on by creating a member function in one of the classes which takes the input parameter based on length, overall utilizing the principles of OOD's.

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.

The objects and how they interact with each other in the function calls, and general program logic of the code is easily readable and interpretable. Variable names are appropriate, variable initializations and any memory allocations are grouped together and well organized with each other. For example, any function calls that require a parameter are easily traceable to what the function is supposed to do. The overall understanding of the function calls and initializations are very quick and easy, not needing to deep dive into understanding how the code works. No negative features regarding the difficulty of interpretation of the main program loop is seen.

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.

Pros:

- Sensible code modularization for better readability and organization compared to C's procedural design (C++ OOD makes it easy to understand where variables are located and their behaviours with member functions)

- Due to the usage of class definitions and objects, debugging is simpler and easier to approach, as each object is isolated within its class. This allows programmers to have an easier time in error location and implementing improvements.

- Flexibility and portable in terms of any changes that may need to be done. Less hard coding is required with the usage of encapsulation and inheritance, and the usage of global variables is close to none.

Cons:

- Requires a deep understanding of classes, objects, and pointers in which it can become very easy to lose oneself in their own code, compared to C's procedural design.

- C++ uses a lot of memory in which many files are created depending on the number of classes required to finish one's task.

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.

In general, the code offers sufficient comments with short and concise explanations of what each component does. However, the case can be made that there were too many comments leading to redundancies. To improve this shortcoming, we recommend having a general comment at the beginning of the function, going over in brief what the function does, as well as commenting in blocks of code instead of every single line. For example, if a block of code is for printing out the snake's body, we would comment at the very beginning of the code block explaining what the code is supposed to do, rather than in every line of the code block.

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.

The code follows good indentation, sensible white spaces, and newline formatting for better readability. It is overall very clean in its organization and does not have any issues. Good indentation is seen in the nested for loops and if statements, while also being paired with curly brackets to outline code block boundaries which shows what statements are within.

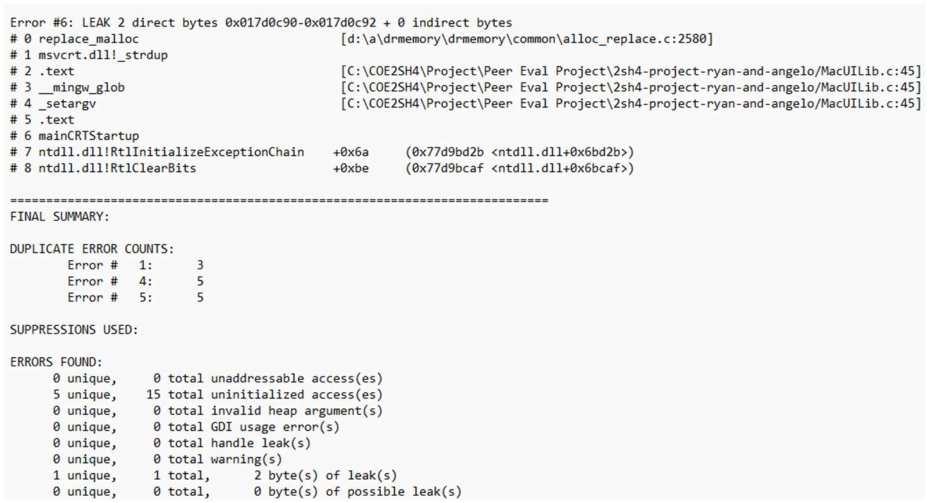
Part III: Quick Functional Evaluation

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)

For most of the playing experience, the snake game does run smoothly as the interaction with the snake head and food flows seamlessly. The wrap around is smooth as well, showing no error in its functionality. Keyboard presses are as smooth as it can get as there is not much delay when pressing the key (not much lag). However, a small inconvenience we have noticed while playing the game was the lack of a proper end/lose game screen. When these actions are executed, the screen does not clear and show a final score screen. Instead, it just stops the code, making it seem as though it has frozen and crashed.

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.

The snake game does not cause any memory leakage as seen through Dr Memory. This is a result of proper memory deallocation of any allocated variables on the heap. MinGW does however sometimes result in 2 bytes of leak if it does cause problems for the person who runs Dr Memory, but the memory leakage has no relation to their code.



```
Error #6: LEAK 2 direct bytes 0x017d0c90-0x017d0c92 + 0 indirect bytes
# 0 replace_malloc                          [d:\a\drmemory\drmemory\common\alloc_replace.c:2580]
# 1 msvcrt.dll!_strdup
# 2 .text                                   [C:\COE2SH4\Project\Peer Eval Project\2sh4-project-ryan-and-angelo/MacUILib.c:45]
# 3 __mingw_glob                            [C:\COE2SH4\Project\Peer Eval Project\2sh4-project-ryan-and-angelo/MacUILib.c:45]
# 4 _setargv                                [C:\COE2SH4\Project\Peer Eval Project\2sh4-project-ryan-and-angelo/MacUILib.c:45]
# 5 .text
# 6 mainCRTStartup
# 7 ntdll.dll!RtlInitializeExceptionChain    +0x6a     (0x77d9bd2b <ntdll.dll+0x6bd2b>)
# 8 ntdll.dll!RtlClearBits                   +0xbe     (0x77d9bcaf <ntdll.dll+0x6bcaf>)

===========================================================================
FINAL SUMMARY:

DUPLICATE ERROR COUNTS:
        Error #   1:      3
        Error #   4:      5
        Error #   5:      5

SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      5 unique,    15 total uninitialized access(es)
      0 unique,     0 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      1 unique,     1 total,      2 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
```

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.

This project was a great experience and provided a strong basis in learning the principles of C and C++. We both may have had a stressful ending during project as the bonus started breaking the code, but at the end we got it to work as expected. One inconvenience was the lack of real time collaboration on the same interface, much like how *Google Docs/Microsoft Word* works. This causes potential clash of ideas when working on the same component, and also having to learn what the other team member has coded, rather than fully working together and coding the components together.