COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Winston Ho (how20) _____

Team Members Evaluated Abigail Phillip (philia10), Krishna Bhatt (bhattk25)

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.

Looking at the header files (objPosArrary.h, GameMechs.h, Player.h, and SnakeFood.h), it is easy to interpret the behaviour of each object as all defined functions have a clear purpose and sensible parameters and return values. Additionally, as the objects were implemented as according to the recommendations in the manual, the code is easy to understand and would be easy to understand by any other person if they had read the manual, which is good as standardization is always good. On top of this, functions/methods that operate on the same variable (i.e. getInput(). setInput()) have been grouped together, reducing the time required to understand the workings of the code.

Although no comments were deployed here, this is not a relevant downside as care was taken to make the code as self-documenting as possible described above, rendering comments nearly pointless.

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.

In the main program loop, it is easy to see how the objects interact with each other, as both global and local variables are named with sensible names in order to describe themselves. Additionally, as a lot of code is hidden within the functions/methods of each object, it is easy to see how they interact with each other as a lot of the noise has been hidden away, thanks to the team's extensive use of OOD.

However, one negative is that the team does not deploy comments in large sections of code – particularly in DrawScreen(). Although this is acceptable in smaller pieces of code present elsewhere in the program as they have taken the time to make sure each function/variable makes sense, in larger pieces of code such as DrawScreen(), it is difficult but after some time possible to figure out the individual purposes of each variable/loop. In the future, in large pieces of code, I would suggest that the team make sure to comment the purposes of for loops and/or some more complicated statements.

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:

- More organized
- Less gigantic chunks of code which are hard to debug and understand easily
- Enhanced readability within classes, and their respective methods/functions
- No accidental modifications due to setters/getters
- Additional protections against unexpected behaviour (such as std::out_of_range) are easier to protect against

Cons:

- Memory management is more difficult as it is more of a requirement in OOD
- Additional overhead to create classes and manage them properly

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.

The code conveys how it operates via self-documenting coding style, which is a benefit as there is reduced need for comments explaining what each block of code does. The team have placed functions/methods that operate on the same variable (i.e. getInput(). setInput()) together inside classes, helping reduce overhead in trying to understand the code. Additionally, they have opted to put the majority of their code within the functions/methods inside of the classes, which help significantly in prevent huge blocks of unreadable code from occurring.

However, the only downside is that they have not compartmentalized DrawScreen(), which is to be expected. This understandably results in a huge block of code, which is difficult to break up easily. However, they have not commented this function, and as a result, it is difficult to understand what the code does at a glance. In the future, I'd recommend that if the team has any large chunks of code, to comment some parts of it to explain the overall functionality of the code block to increase readability.

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.

Yes, the team has formatted their code with standard white spaces, newlines, and indentation which allows for good readability. This formatting is flawless across all of their header and source files, which is great and allows for excellent readability. The only minor nitpick is a misindented bracket at Project.cpp:86, however this again is a nitpick and the team has been flawless with their readability in all other areas of the code.

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 the most part, the Snake Game offers a smooth, bug free experience. Food collection, food generation, snake collision, scoring, lose conditions, are all implemented correctly. However, there is a minor bug inside of the code.

The bug is when the lose condition is activated from a collision, the final score and lose message is not displayed to the player, as the team have not implemented this. In order to finish this task, the team should add a MacUILib_printf() call at Project.cpp:142 in between clarScreen() and uninit() inside CleanUp() to display a message after the screen has been cleared and before the library is deactivated.

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.

No, after running DrMemory, the Snake Game does not cause any memory leak. The only memory leak was from typical sources such as MacUILib's getChar(). The team has been careful with potential memory leakage and has ensure that the properly allocate and deallocate any pointers that they use though their code. They have made sure to deallocate within CleanUp() and their destructors.

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.

The project went well for the most part, however having a partner would've helped as this project contained a lot of code, more than was typical in the PPAs. Having a second pair of eyes would've helped when tracking down potential issues, and ensuring each criteria necessary for the Snake Game could've been met. Although this project is certainly possible solo, a second person would definitely reduce the amount of time necessary to complete the project and make the time necessary to complete the project much more manageable.