COMPENG 2SH4 Project – Peer Evaluation

Your Team Members <u>Abigail Philip - philia10 + Krishna Bhatt - bhattk25</u>

Team Members Evaluated <u>Winston Ho - how20</u>

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.

Observing the header files of each of the objects, it can be said that they are very nicely structured and grouped within hence leading to ease in understanding the functionality. For example, in the Player.h file, there are comments above some of the functions which explain the function of some of the more complex ones. The individual has also chosen to create the 'Food' functions within the GameMechs.h files and separated them properly so the functions can at first glance be assumed to work with each other or regarding the same functionality. Parameters with the functions are also named accurately helping understand functionality.

However, there is still room for enhancement. In regard to the Player.h file, although there are comments explaining functionality, the way each comment is structured can be made better for higher readability. There are also some functions present, e.g checkForFutureCollision and checkIfSelfCollision which seem different than those present in the manual and the function can likely be integrated into GameMechs and/or other existing functions in the Player file. Integrating these functions into existing functions within the Player file or relocating them to the GameMechs file can make the code more coherent.

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.

Examining the main logic in the main program loop is relatively straightforward and enables the interaction between GameMechs, Player, and other classes. The modular structure properly separates concerns and the use of functions for each of the sections. Also, it clearly shows a sequence of events as well as properly defines conditions within the while loop using pointers. The functions are seemingly properly called from their respective class. There are sufficient comments explaining the functionality as well as previous iterations. Overall, it is quite simple to interpret with little time.

Though as it pertains to negative aspects, there are very few as the code is properly readable as well as properly commented. One flaw could potentially be keeping older iterations present in the code by only commenting it out as this can make it look messy. However, this has no impact on functionality.

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 :

- Code is organized into classes, which aids in separation, easier management, and makes it easier to identify issues
- Objects can be reused in different parts of the program without having to repeat lengthy code
- Helps with code readability by structuring and separating the logic

Cons :

- Requires careful memory management with pointers
- Sometimes harder to apply as multiple classes may need to be implemented into each other

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.

The code in the Project.cpp file has enough commenting where most features are properly explained and the logic is easy to follow. Though commenting is lacking in most if not all other files, especially the .h files. Comments at the beginning of each class definition as well as within would provide an overview of the class purpose and dissect steps taken to achieve this purpose. There are also commented-out sections and variables that could have been used to better explain why it is commented out and how it was fixed to show the design process, eliminating some present ifs and buts. Though, that being said, at what the code is, the commenting and self-documenting are detailed enough to help understand the purpose of functions as well as specific blocks in the code.

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.

The code generally follows decent indentation, white space, and formatting conventions. The indentation is mostly consistent except for a few places. One spot could be the getInput function where after the Boolean, everything is shifted slightly right. Also, the usage of while spaces to separate logical sections within functions and between different sections of the code is generally good. Though regarding the alignment of the comments, there are places that need improvement. The comments in some places are not consistently aligned with the code they describe. For example, the comments above the objPosArrayList::find, can confuse as some of the variables described are parameters while others are local variables, but they aren't in order. To fix this, next time perhaps create comments with wanted actions within a function then create code pertaining to it under the comment, and then move on to the next. For important variables, comment directly beside or above with a brief explanation.

Part III: Quick Functional Evaluation

[6 marks] Does the Snake Game offer smooth, bug-free playing experience?

1. 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)

There are a few major logic errors within this code that keep it from performing the tasks it's supposed to. Although the same game seems smoothly running for the first few food collections, once the snake gets long enough, one will find that the self-collision logic is not implemented correctly. Currently, even if the snake hits itself, the game continues, making it essentially never-ending and causing semantic bugs in the code. To address this, it is recommended to debug the checkIfSelfCollision function and thoroughly review its implementations in the runLogic function. Debugging techniques, such as using breakpoints or printing variable values within the original function, can help identify and resolve any loop errors. Other than that, other features work well. Majorly, the food gets collected, making the snake grow, and the score updates. The wraparound logic and snake movement direction are also functioning as intended.

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.

The Snake Game code does not cause a memory leak. In the CleanUp function, both pointers, game and player, are deleted, ensuring that there is no memory leak carried over to the next game. The code uses ‘new’ for dynamic memory allocation and ‘delete’ for deallocation. Additionally, destructors are implemented where necessary to ensure proper memory deallocation.

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.

Working in a collaborative software development project was a new experience which presented itself to be a major learning experience. With the initial delegations of partners A and B aided significantly within the process as it allowed both partners to work independently without needing the others code to implement their own. This made it helpful to work at one's own pace with a decided personal deadline. Though one thing that presented as a learning opportunity was when in iteration 3 we both had to use each others created code and integrate the classes together. This required a significant amount of communication.