COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Tina Gao & Chloe Cheung

Team Members Evaluated Shen & Meng

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.

The header files seem pretty organized and is easy to understand what each function may do. However, there are an excessive amount of functions in Game Mechanics, such as the getKey() functions, that at a first glance you can't predict its interactions with other functions and classes. Otherwise, the methods are logically grouped with spacing and appropriate comments describing the similar functions. For example, methods dealing with exit flags are grouped together. Skeleton code comments were also left in header files throughout. The food class that should have accompanied the bonus was also missing, though its functions were appropriately defined as a part of the game mechanics class instead.

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.

How the objects interact with each other can be easily interpreted in the main logic except in the draw screen function. There is a lot of code before anything that relates to displaying things to the user. Those functions/logic should have been placed in game mechanics since they relate to how the game should react to the snake picking up certain foods. In the run logic, there is also mention of a win flag but, there should be no win condition for this game. Overall, the spacing and indentation provides good code readability, and there are comments throughout. More descriptive comments could have been provided to describe the logic throughout the program and in class files.

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.

The pros of the C++ OOD approach is that after you design classes for objects in the game, you can focus purely on how those object classes interact in functions, then appropriately call the functions in the main program without all the lengthy accompanying code. This allows for coders to "work backwards" when designing new features to add on and they would already have many sub-functions to work with from the objects' classes. The major con is that if there is a behavioural bug, it could be a nightmare to debug since many classes could be interacting and the location of the bug is unknown.

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.

Overall, the code lacks immensely in comments. Those that are included are very high level and there are minimal comments in the class function definitions. To improve this aspect, comments could be added for most functions to describe what they do (if the function is not obvious by the name). For example, the check food consumption function in the player class should have comments to explain the logic and comments to explain the outcomes of the different foods that are consumed. The included txt file to explain the features of the bonus was useful though those explanations could have instead been included in the .cpp files.

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.

Overall, the code has good readability using indentation, whitespaces, and newlines. It was slightly lacking in the player class where many if and if-else statements are stacked, but the indentation helped with that readability.

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)

The game offers a smooth experience playing it though the implemented bonus feature that speeds up the game occasionally makes the game refresh too fast and hard to play. Otherwise, there doesn't seem to be any apparent bugs while playing the game.

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.

There is no memory leak when running the Project.exe through drmemory.

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.

In the first 2 iterations, both partners could work at the same time without many git pull/push issues. There was sufficient time to collaborate to finish each iteration. The manual also well-described the requirements of each class. The biggest obstacle was git wanting to abort a git pull when any of our local repositories didn't match the main git repository which we didn't know how to get around. If we did this project again, we would meet after each iteration to explain the classes we worked on so anyon e working on subsequent iterations knew how to use the class funcions.