COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Ashviya Jeyaseelan & Loretta Lau

Team Members Evaluated Carson & Luca

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.

Overall, the header files of each object are very organized and given appropriate names. The method names are intuitive and human-readable which makes them easy to follow. It is easy to identify how methods interact with each other because the naming scheme serves as a straightforward representation of what the method is supposed to perform. For example, in the GameMechs class, not only are the methods grouped by relevance, but methods like getExitFlagStatus() and setExitTrue() makes it clear that these methods are interacting with each other to determine the status of the exit flag. Since the getter and setter related to the exit flag are named as such, their purpose and relation with each other can be easily understood. The member variables are also name in a similar manner where their purpose can be easily determined without looking elsewhere. For instance in the GameMechs class, boardSizeX and boardSizeY are self-explanatory and it can be understood from a glance that these variables represent the x and y dimension of the gameboard.

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.

Based on the main logic in the main program logic loop, we can easily interpret how the objects interact with each other in the project logic. There are several positive features that make the program easily to follow, such as the naming scheme. The function names within the main program loop are straightforward and represent the actions it completes in the main logic. For example, based on the name the group has selected for the “GetInput()” function, it is apparent that this function obtains the input from the user to be used in the game. It is also easy to understand how the objects interact with one another because they are initialized at the beginning of the project file. When used throughout the project, the variable names that are selected allow us to easily understand which objects are being used and why. For example, the group has initialized an object called myPlayer that calls upon the Player class, allowing us to easily interpret how the object interacts within the program logic and which methods are being used. Furthermore, the function prototypes are placed in the order of execution, making it intuitive and easy to interpret the sequence of actions that are executed in the game. It is understood

that the program first initializes, then obtains the input, runs the game logic, displays the screen, and finally clears the screen and cleans up once it exits out of the game loop.

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:

- The C++ OOD approach in the project allows for a more compact and easier to understand program as all the related functions are grouped together in separate files and are only referenced when necessary. This in turn allows for more complex programs to be created.
- C++ OOD involves inheritance which allows for fields and methods to be public and private, providing more security than the C procedural design approach where there is no protection or hiding of data.
- Due to the use of objects and separate files, code reusability is available which allows new programs to be built more efficiently rather than having to build them entirely from scratch. Features can be edited, added or removed to the designer's liking, saving time and resources.

CONS:

- It is important to keep track where all the methods used are located. Since each class must be referenced and the memory needs to be allocated before being able to call the appropriate method, it can be confusing when trying to remember where the method needed is located.
- When working with programs with several classes, it can become hard to keep track of all the features in comparison to C procedural design, which involves top-down programming and can be preferable based on the objective of a given project. C procedural design may be better suited for more simple programs.
- It is important to be aware of any possible memory leaks in the program. When using the OOD approach, it is important to always remember to allocate and deallocate memory to prevent any memory leakage. This would not be necessary when using the C procedural design approach.

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 comments included are descriptive and helps us understand the functionality of the code more efficiently. For example, the comments in the Project file clearly describes what each line of code is responsible for. When looking in the DrawScreen() function, the comments can be used to understand the purpose of each for loop and what each index refers to. The program also deploys a sufficient self-documenting coding style in addition to the commenting, where human-readable names that are easy to understand and follow are consistently used. For example, in the DrawScreen() function, easily readable names are used, making it easy to understand how the player symbol and the board frame is being placed on the screen, as well the food symbols. When "playerBody" is used, for instance, it is clear the loop is iterating through the list containing all the players in the snake.

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, has sensible white spaces, and deploys newline formatting for better readability. When indentations are used, there are also {} used which makes the code more legible and a lot easier to follow. White spaces are sensible and used appropriately. Newline formatting can be seen throughout the project files as most functions/methods are separated by a new line. However, there are some inconsistencies where a new line is not included between different functions/methods which can be seen in the Player.cpp file between checkFoodConsumptionSpecial() and checkSelfCollision() as well as in the Project.cpp file.

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 Snake Game offers a smooth, bug-free playing experience. The program can print the game board and the symbols, as well as execute the snake food growth consumption and random food generation. The program also prints out the score and guidelines that specify how many points can be obtained for each type of food. Although the program executes smoothly, there are minor edits that can be made to improve the playing experience. Firstly, the instructions on how to play the game may be unclear for first-time players, so we would suggest printing brief instructions under the board. The key used to exit the game was unclear as well, so we recommend adding exit instructions as well. The program can successfully display the self-suicide message, but it does not print out the exit message despite being implemented in the DrawScreen() function, suggesting that there may be an issue. This is likely a result of the program implementing the CleanUp() function and immediately clearing the screen after the exit key, the spacebar, is clicked. To debug this, we would recommend using the debugger with breakpoints and verifying what the program does immediately after the space bar is clicked, specifically where it sets the exit flag to true and exits out of the while loop in the main program function. If it is a result of the screen being cleared immediately after the exit key is hit, we would suggest moving the conditional statements that display the self-suicide and exit messages after "MacUILib_clearScreen()" is called in the CleanUp() function. This will allow for the exit message to be displayed after the screen is cleared.

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 leak according to the memory profiling report. However, at the beginning of the report, it indicates that there are cases of unaddressable access of freed memory. This is likely due to deallocation occurring without having allocated memory in the first place. For instance, in the Player file, "playerPosList" is being allocated memory with the 'new' keyword within the constructor and then deallocated with the 'delete' keyword in the destructor. However,

“mainGameMechsRef” and “foodInfo” are being deallocated as well in the destructor without having been allocated memory with the ‘new’ keyword earlier. This can also be seen in the Food file, where “mechInfo” is deallocated with the ‘delete’ keyword in the destructor without having memory allocated for it. To ensure that memory is being allocated and deallocated properly, it is important to verify that the ‘new’ keyword followed by the ‘delete’ keyword in the destructor is being used correctly

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.

Our first collaborative software development experience was a very pleasant one (hopefully). The thing that made the entire project very enjoyable was because the constant communication we had throughout the development process. When there were certain aspects that one of us didn't understand, we could ask each other and discuss possible solutions together which made the debugging process a little less tedious. On the other hand, working with a partner on this project was difficult because there were times when we didn't understand how the other half of the code worked without a thorough explanation from the other partner. When combining the player features and game mechanics, it was hard in the beginning when trying to understand enough of the other partner's code to implement it together.