COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Raymond Pham	Arabella Paet
Team Members Evaluated	Raymond Pham	Arabella Paet

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

Note: The group that we were assigned to did not finish the project therefore we decided to evaluate our own project.

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.

   GameMechs.h
   - objPos
   - objPosArrayList
   - MacUILib

   The game mechanics class was easily interpretable as methods of similar functions were grouped together and commented. The method names accurately describe what each of them does. It was clear that the class dealt with aspects of the game and its rules. One thing that could be improved is the redundancy of the methods. A lot of the methods are concerned with setting a specific parameter true or false. To improve the conciseness of the code, these methods could be put into one setter method that can set an input parameter true or false based on another input parameter.

   Player.h includes:

   - GameMechs.h
   - objPos.h
   - objPosArrayList.h
   - Food.h

   Based on the methods within the Player class, one can infer that the class controls how the users object moves and interacts with the game board. It interacts with the GameMechs class and food class as there are used as parameters for its constructor. The Player class can also be seen to interact with objects from the objPos class for parameters of the methods created such as checkSelfCollision(objPos head) where one can infer that it takes in a objPos object and checks if the Player object has collided with that object. Since C++ has the capability of function overloading, checkFoodConsumption(), and checkSuperFoodConsumption(), could have used the

same function name but with different parameters. Overall, the Player class seems to exploit OOD well with it interacting with multiple different objects from different classes.

Food.h includes:

- objPos.h
- objPosArrayList.h
- GameMechs.h
- MacUILib.h

The food class can be inferred to control food generation and related functions. The methods are organized in a way that setters, getters, and constructors/destructors are grouped together and commented their specific function. Something to improve on is to comment on what the non-standard methods do like add() and isExist(). Also notice that MacUILib is included but it does not seem to be used. Including unnecessary libraries could affect the program efficiency or lag the program so it is important to check and clean for code efficiency.

objPosArrayList.h includes:

- objPos.h

Looking at the methods in the objPosArrayList, we can infer that the class works with the player snake body, specifically the snake body elements. The function of each method can be clearly assumed from the method name, and it is clear what the input parameter is being used for. The methods with similar functions are grouped together and commented. One thing to improve on is to comment what the non-standard methods do like the emptyList method. It would help to comment what it does or why we need it for.

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 RunLogic() in the main program loop is very concise which makes it easy to interpret how the classes and objects are used within the project. Within the main logic, the ptrPlayer object calls onto methods from the Player class: updatePlayerDir() and movePlayer(). One can infer that these methods called by the object controls the player movement based on the users input and on the intuitive names of the methods. At the end of run logic, the ptrGameMechs object calls on the clearInput() method which one can infer, clears the users input from the input stream. Overall, due to the conciseness of the main logic and the names of the methods that the objects are calling, one can interpret the functionality of what is happening within the code block.

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:
   - There are less global variables written in static storage
   - Since there are less global variables, the total memory storage on the heap is maximized
   - less floating variables
   - improved code readability and conciseness (didn't have to keep scrolling up and down a large file to find what a specific variable does)
   - Better user readability (The main program makes intuitive sense when a coder reads it)
   - Very modular since there are different classes that hold specific methods to do certain things instead of a list of function definitions at the end of the program (in PPA3)

   Cons:
   - More files to deal with instead of 1 large file
   - We have to keep track of access level between classes (What is made private/public to outside the class)
   - It was a little more difficult to get a grasp on OOD for C++ than C procedural design in PPA3
   - OOD affects memory performance more than C procedural design as creating objects takes up a lot of memory.

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 has a good number of comments that effectively explain portions of the code where it is not immediately obvious what is taking place. The logic for various code blocks was explained in the comments which made it easier to understand how various data was processed. The sufficient commenting also ensured that it is easier for both members of the team to work with the other person's code without having to go through the whole document on each class and method. Something to improve on would be the redundancy of the comments. For example, in the GetInput() function, there is no need to comment what that function does as it is obvious from the function name. Unnecessary comments could clutter the code instead of making it simpler to digest. It is important to keep the comments concise to ensure that it does not negatively affect code 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.

   - The program follows good indentation with code blocks within function definitions and for/while loops are indented for better user readability. New line characters were also printed after each line within the DrawScreen() function inside main, again for better user

readability. Some singular lines of code are quite lengthy as it goes off the screen when a user to trying to read it. For instance, in the DrawScreen() function in main, there is an if statement that checks if the coordinate is on the edge of the board. The if statement has multiple conditions all written onto a singular line. To improve readability, a newline could have been inserted to fit the if statement onto 2 lines instead of 1 continues long line. Appropriate white spaces between characters can be seen within parameters of functions, loops, conditional statements, etc. It helps readability as the code is bunched up together, but instead consists of proper spacing just like a regular sentence should be.

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 executes and correctly displays the starting screen with the player object in the middle of the screen and the border printed correctly. The food objects are generated at random locations each time one is eaten, and the snake body and player score are incremented based on the food eaten. The snake dies when eating its own body which means the self- collision function is implemented correctly. Proper lose, win, and exit messages are printed depending on the situation. The program flickers quite a bit but this is probably due to the print delay. Some things to improve in terms of user experience is to display some necessary info on the screen. It might not be immediately obvious to the user how to exit the program or what happens when you eat different food objects. To improve this, print statements on important info for the user/player like the exit key, what the superfood does, and rules of 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.

   When running drmemory on the snake game, no leaks/possible leaks were recorded. Proper deallocation was done within the CleanUp() routine in the main function for anything that was created onto the heap during the Initialize() routine. Destructors for classes were also properly implement to delete anything on the heap that the constructor or methods from the class may have created onto the heap.

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.

    - It was difficult at first to understand how to collaboratively work together on a single project. However, with the use of OOD we were able to work on specific portions of the project parallelly by creating different classes/libraries and then combining it in the end. We learned that communication is key when collaborating on a program. The use of comments was very helpful as it allowed each other to understand our assigned coding sections without to having to constantly asking each other what a specific piece of code does.