COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Dev Patel (pated158) & Rajveer Sandhu (sandhr22)

Team Members Evaluated: Viraj (banev) and Tai (li1568)

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.

Evaluation: Looking at each of the header files, the functions were all defined with appropriate names that easily describe what each method was intended for. Likewise, it appears that all data members were used with the correct implementation of encapsulation, considering that all variables were defined in the private scope. Moreover, it is clear how every object interacts with one another, given that the objects instantiated in each header file are used for carrying out the operations needed in each method. For instance, in the Player source code, in the method movePlayer(), they instantiate 2 objects representing player head and food of type objPos. These objects were then compared in their method compareCOORD() to check whether the objects overlap in position, thus checking for a collision. This use of object interaction is a good solution for how the program can check if the player has eaten food. Overall, the header file implementations show sensible use of OOD.

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.

Evaluation: In the main program loop, it is simple to interpret how the objects interact with each other. For example, they instantiate an object from both the GameMechs class and the objPosArrayList class, using the latter to create a 2D List of the gameboard as an objPosArrayList object. Additionally, they created a GameMechs pointer that the objPosArrayList object uses for getting the gameboard dimensions. Conversely, from a memory standpoint, creating a 2D List for the gameboard, rather than printing the gameboard elements character-by-character, is not the most memory-efficient. Furthermore, when calling the DrawScreen() function to access the positions of the food and player objects, they call upon their respective methods for getting the objPosArrayList. Consequently, the gameboard can print out all food and player body segments in the correct positions. All objects are used effectively in the main program so that it executes as expected and appropriately invokes other classes only when necessary.

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 interactions between different objects, thus they all need contained data members with specific methods to execute their tasks effectively.
    - Better code readability, considering the larger scope of the project.
    - Simpler to maintain and debug.
    - Allows for encapsulation of data members, ensuring they are not modified by objects not associated with them.
- Cons:
    - Unnecessary and excess classes for smaller to medium sized projects.
    - More memory intensive with more objects (thus data members) allocated.
    - More susceptible to dangling pointers or unexpected changes in object data.

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.

Evaluation: Predominantly, the variable and method names are straightforward enough to indicate their purpose. Consequently, the source code is by and large self-documented. However, to further highlight the purpose of the methods (aside from getters and setters), there are many instances where there are comments directly above the method definitions that summarize what each function does. Conversely, the documentation could be improved by providing more direct comments that explain what is being implemented after every few related blocks of code.

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.

Evaluation: There is good readability with all the source code files when concerning the use of indentation and whitespaces within conditional statements. However, there are some shortcomings, as there is insufficient use of newline spacing. This is seen in the Player source code, where several conditional statements are lacking in newline formatting, such as in lines 301 and 305, where there are back-to-back conditional statements that touch the curly braces of the previous conditional statement. This along with several other cases that repeat this same lack of newline formatting, reduces the readability of the code. Nevertheless, the overall code is understandable and has adequate 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)

Evaluation: Most features in the gameplay are flawless. These include a seamless wraparound of the player body and good implementation of the collision logic. In particular, for the collision logic, we see quick and correct invocation for the generation of new food objects, along with the changes in the player body and player score. Moreover, the end-game mechanism occurs directly once the player's head overlaps with other body segments, indicating correct execution of this functionality as well. Although many features exceed expectations, there is one underlying issue in the input collection portion of the code. Upon playing the game, there appeared to be a delay in processing a user input for changing the direction of the snake. After inspecting the code responsible for this functionality, it was noted that in the Project.cpp source code, in the RunLogic() function, the movePlayer() method is called an additional time before changing directions. This occurs as after taking the user input, the movePlayer() method is invoked before the updatePlayerDir() method, making it appear that the snake direction change lags behind the user direction input.

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.

Evaluation: There is only 2 bytes of memory leakage which comes directly from MinGW. Consequently, this memory leakage is not a concern. By analysing the main logic in Project.cpp, we see that all 4 globally initialized object pointers, which are allocated on the heap, are all deallocated in the CleanUp() function. This deallocation occurs by using the delete keyword on all 4 object pointers. Within other classes, such as Food and Player, there are object pointers that are initialized in the constructors and dynamically allocated, and they are always deleted in the destructors. Accordingly, their correct invocation and deletion of object pointers on the heap ensures that the program has no memory leakages (aside from that due to MinGW).

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.

Thoughts: Working on our first collaborative software project together, there were many pros and cons to take away from it. From an advantageous standpoint, there was more creative input and quicker problem solving. As a result, we were able to cover each other's weaknesses and provide solutions when the other hit a roadblock. Conversely, the only issue we faced was there was no real-time collaboration on the code, as we both could not simultaneously modify the code and push it.