COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Hashan Rex & Nikha Jinosh

Team Members Evaluated Alexandros & Ilya

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.

   Looking at the header files of each object, it is visible that the team has used good code modularization in terms of object-oriented design (OOD). A positive feature is the use of getter and setter methods, which are seen throughout all the header files. This allows the programmer to control how the internal state of the object is modified and accessed, which is crucial for encapsulation because it hides the implementation details. Furthermore, the programmers have organized the getter and setter methods depending on their common use. For example, in the GameMechs.h file the methods that depend on the input are in a separate paragraph from the methods that depend on the score. This allowed for easy interpretation of the possible behaviors of the objects in the program. Another positive feature is that each header file has a clear purpose. For example, the purpose of the objPos class is to represent a position and this can be seen through the simplicity of the titles of the methods. This also aided in the understanding of how the objects interacted with each other in the program.

   Even though, the header files were expertly done, a negative feature that could've otherwise aided in the further understanding of the behaviors in the program is the lack of comments. Additional comments would've enhanced the readability of the code and provided more explanation of design choices. For example, the programmers could've commented on which methods were added to each header file by the group versus which ones were already in the file and provided by the professors.

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.

   While examining the main logic in the main program loop, it can be easily interpreted how the objects interact with each other in the program logic through the code. A positive feature is the logical conditions in the main loop (specifically the while loops), which clearly indicate the different states of the game. For example, a while loop is used to check when the maximum score of 364 is reached. Another positive feature is the primary structure of the main program loop, in which the programmers accurately used the skeleton code provided by the professors. The structure itself is easy to follow and can be simplified into 3 steps (getting input, running game logic, and drawing the screen) due to its strong readability. Each of the steps is connected with a function (for example: GetInput) and this aids in the maintainability of the code. The programmers provided comments throughout the program loop, however, a negative feature is the lack of comments during the nested for loops portion of the code. During this part, it's confusing to understand the condition presented by each for loop and how it relates to the overall modularity of the program. Overall, the main logic was nicely executed and represented all the important tasks needed for the assignment.

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:
   - Object-Orientated Design (OOD) promotes modularity, which is the practice of dividing the code into smaller and independent parts that perform specific functions. This allows the code to be more maintainable and by using the C procedural design approach, it makes it harder to maintain the code since it isn't in independent parts.
   - OOD ensures code reusability, in which the classes and objects created in the project can be reused by incorporating them in different areas of the program. This also allows new classes to be built off of old ones, by inheriting specified behaviours. Whereas, in the C procedural design approach, it is more challenging to reuse the code.

   Cons:

   - Understanding OOD is a learning curve for developers because the concept of encapsulation (which is the basis for OOD) is confusing for people who are mainly used to global variables and functions.

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 follows a style that is quite sufficient in self-documentation, and in areas where needed comments are placed, otherwise it's easy to understand, with the exception of two small areas. In most areas where there is code that could be confusing without context, there are comments indicating what the statements do, an example being the main function of the Project.cpp file. The comments in the main function helps understand the how the different game conditions work, without comments here it could be a bit confusing, but it is still quite straightforward. However, one area that could use some comments could be the for loops area in the DrawScreen function. Some additional comments could help with the understanding of how the things are drawn on the screen with the use of the loops. Additionally, some extra comments in the header files would be good for those who are not familiar with the program (e.g. explaining the loseFlag).

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.

   Throughout all the files, the code follows good practices which helped the code have good readability. The code has indentations where needed to help with the readability of functions, conditional statements, and loops. There is also good use of newline formatting for overall better readability. The code also has good use of white spaces to help differentiate between the different features that are in the same method, as seen in the movePlayer method in Player.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 and bug-free experience. After a couple of runs of the game, no buggy features can be observed. The snake moves as it should, it cannot go in the wrong direction (e.g. switching directions from left to right) and when it collides with itself the game ends as it should. The generate food and snake growth mechanisms seem to be working correctly, as you can see the snake grow and food being generated again. The wraparound mechanism is also working properly, and the game boarder is static how it should be. Also, you can see the difference between the player quitting out of the game versus losing the game, and when you lose your score is also displayed. Overall, the snake game was working flawlessly with no bugs to be found.

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 memory leakage. The drmemory report states that 0 bytes are leaked, and based on the code, the game is coded correctly to not cause leakage. In the Project.cpp file, two pointers were assigned to the heap in the global scope, and at the end in the CleanUp routine the pointers were deallocated correctly. Also, throughout the class files the constructors and destructors were used correctly, as seen in both the Player and objPosArrayList files.

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 collaborated software development has been an amazing learning experience because unlike the previous labs and PPAs that we completed for COMPENG 2SH4, we were able to rely on each other for help and further understanding. Instead of feeling alone and confused, we reached out to each other to see if we could combine our knowledge and complete the task. The only thing that was proven to be difficult in this experience was using "Git Pull" and "Git Push" continuously when working on the program. Sometimes GitHub lagged and it was hard to continuously split the work without changing something that the other person has worked on.