COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Aadeeba Jaima	Marryam Kamal
Team Members Evaluated	Chehan Marakawatte	Allie Launder

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.**

   Yes, we can interpret the possible behaviours of the objects. They have good function names that are self-explanatory, but there are no explicit comments on what each function does. In the future, consider adding comments to make the function of each method in a class easier to understand for the viewer.

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.**

   Great use of comments, it helps follow the program flow. Great code efficiency in the RunLogic function through the creation of extra functions in the Player class, but a disadvantage of hiding most of the logic in the player class is that it makes it harder to follow the logic and observe how the different objects interact with each other. The RunLogic function states you are moving the player but does not show what happens when there are collisions with the food.

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:

   - Simplified logic in main program file
   - Shorter main program file
   - Reduces code repetition
   - More explicit representation of what each algorithm does in the code because it's all put under a method with one name
   - Allows for the creation of multiple instances of the same object type so we don't have to type out the same logic each time for new instances

   Cons:

- More coding and pre-planning and code organization required to ensure the different class interact properly and are created correctly
- Multiple files
- The use of classes within classes becomes difficult to keep track of
- Increased memory usage

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 main file Project.cpp could use more comments in the RunLogic function especially since so much of the logic is happening inside of the player class which makes it hard to follow the logic. So, comments to explain more in depth what is happening in the RunLogic would help someone who's never seen this code understand. All the classes contain decent comments. But the player class could be improved. There are a lot of places with long one-line comments and no lines of comments in the actual function. Overall, the comments are pretty good and explains the code well.

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 and newline formatting to separate different blocks of code. One thing to improve on would be to add more whitespace characters, especially in for loops and if statements, so the code seems less cluttered and easier to read.

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)**

   Overall, the gameplay is smooth, and all the required functions work properly. However, a bug we observed was that the snake appeared to freeze for a moment each time it collected food. A possible root cause for this may be the way they implemented their algorithm to make the snake grow in length each time it collided with the food object. In their player class, it appears they implemented food collision checks in the same method that controls the player movement direction using an if condition. This means that it will either move, or increase player length when food is collected. Because these two actions don't happen simultaneously, the snake appears to freeze before continuing its movements. In order to fix this, we propose that movement and food consumption checks be placed outside an if-condition (and not in the same method) so they're able to run simultaneously and not result in random freezing.

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.

   No, Dr memory reports 0 bytes of leaked memory in their program.

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.**

Overall, all it was an enjoyable and educational experience. Splitting the tasks allowed us to finish it faster. We each had our own sections to focus on which allowed us to become very familiar with our own parts. A con was that since the tasks were slip and each partner only worked on certain parts of the code, when we ran into bugs it was harder to overcome the bugs alone and took some time because both partners had to collaborate to overcome the errors. But overall, it was a rewarding experience.