COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Everson Shen, Mohamed Ali

Team Members Evaluated Andy Ngo, Vishva Madu

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 easily predict the behaviours of each object. The members are appropriately named, making it easy to identify their purpose. They thoughtfully grouped the setters and getters, enhancing the overall code readability, and making it easy to follow their logic.

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.

Yes, we can follow their logic and understand why and how each object interacts with each other. In their project.cpp file we liked how they initialized everything in an organised manner, because it enhances the overall readability of their code. Though we understand why they used the time.h library, it was unnecessary for the random position generation and made it slightly less efficient.

A positive feature of theirs was using a reference of *GameMechs* through the class *myGM,* while also having an object for player position and food position, two very smart decisions. Modularity was very good in this class from their use of objects and provided abstraction, making the code more readable and understandable. Observing the code, we see a few global objects which is smart for our project as these objects will need to be seen everywhere in the function. Ideally of course, we would want to reduce the number of global variables and objects, highlighting a potential negative feature. Overall, the code used object-oriented design well.

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.

- OOD Design:

    - Pros:
        - More effective for larger scale projects because it reduces redundancy
        - Increases organisation and structure of code, making it easier to navigate and locate specific features
    - Cons:
        - It's a new concept so it was relatively difficult and unintuitive to implement.
        - Ensuring correct initialization can be time-consuming, and navigating across multiple files might pose a challenge for developers.
- PPA3 (procedural programming):
    - Pros:
        - It's more straightforward and intuitive to implement, so carrying out tasks can be done more easily by the programmer.
        - This is most efficient and quickest to implement for smaller scale projects like in the PPA's because there is minimal initialization
    - Cons:
        - It's easier to debug and find issues because everything's in one place
        - Less suitable for larger projects due to potential difficulties in maintaining code organization and managing complexity.

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.

While the code does not offer sufficient comments, the code deploys sufficient self-documenting style quite well helping me to understand the code functionality well. I can decipher functions in their classes and the *project.cpp* file well. Despite this, the *project.cpp* main file can feel a little cluttered and it might be difficult to understand to other people. The class functions are much more organized than the main, so we would recommend providing comments for in-loop conditional statements and workflow.

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 does follow good indentation, especially for loops which make conditional statements very easy to go through. There is a decent number of sensible white spaces, but we would personally like a little more for an easier time reviewing, mostly in the *drawscreen* and *runlogic* functions. Another point we would like to point out is the consistency of curly brackets. Spacing is always the same and the bracketing is always in chronological looping order on separate lines and columns.

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 snake game offers a smooth experience with a small noticeable bug, which is noticeable upon eating food. Upon inspection, the snake grows when eating food but not instantly. The score goes up instantly when food is eaten, however the snake length increase appears to happen in the middle of the snake and not immediately at the head of the snake. A probable root cause could be from the *Player.cpp* file, *Player::movePlayer()* function, where we call the *insertHead(currentHead)* call and *removeTail()* function. The reason for this is because the snake is not inserting the head correctly when eating a food, which could be from the food consumption function. we would recommend to first evaluate the function that checks for food consumption, and the algorithm used to implement a list element to the head and tail. Then, we would recommend evaluating how the *insertHead, removeTail,* and the food consumption function interact with each other during food consumption.

A slight modification we would suggest is to create a separate *loseFlag* and *exitFlag* as well. Currently, when the snake suicides, there is a different *loseFlag* message, but the program does not terminate immediately. You would have to press the exit input (space) after seeing the *loseFlag*. This is not a big mistake, but this differs from the *SampleExecutable* given. A possible implementation of this is to set the loseFlag and set the exit flag to true consecutively in a conditional statement.

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 leak, indicating proper use of destructors in the object classes.

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 experience was overall fine. It was a little difficult working separately due to overwriting commits, which ultimately improved after a learning curve with github, (learning how to use git stash, git stash pop, etc). Communication was very important which is something we upheld very often. In general, everything worked fine and nothing was done poorly. However, for other teams we do recognize that this project could have been very difficult to do collaboratively if communication was not established well from the beginning. For example, overwriting commits would happen more often.