COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Mohan & Rehan

Team Members Evaluated Umar & Hamza

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.

   ### Positive Aspects:

   The header files (e.g., objPosArrayList.h, Player.h, Food.h, GameMechs.h) define clear interfaces for each class, indicating a good level of modularization.

   The separation of concerns is probably evident, with each class handling specific functionalities (e.g., Player for player-related actions, Food for food mechanics).

   ### Areas for Improvement:

   Some functions may be not needed (I.e., checkIndexVlidity() in objPosArrayList.h), as it is more like a debugging function, which might lead to less encapsulation.

   Effective use of comments and documentation within header files can greatly enhance understandability, especially for future maintenance or extension of the code.

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.

   ### Positive Aspects:

   The main program arranges the interactions between various objects effectively, demonstrating good OOD practices, with reasonable pointer definition, reference usage.
   Object interactions in GameMechs.cpp and Player.cpp is well-structured, facilitating easy understanding of game mechanics.
   Comments on every step in the main loop are provided nicely.

   ### Areas for Improvement:

Aside from the instruction provided for the project, if the UI design could be written out of the main program, it will be more encapsulation and clearer to check the main code for debugging usage.
The current implementation doesn't prominently feature error handling or exception management.

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 of C++ OOD:

Enhanced code organization through classes and objects.

Improved maintainability.

Facilitates more complex and versatile software design.

The objects are effectively dependent on each other, demonstrating a well-implemented Observer pattern where changes in one object affect others.

Effective management of dynamic resources (heap-allocated objects) is shown, which is crucial in C++ for preventing memory leaks.

Cons of C++ OOD:

Difficulty of learning C++ OOD system.

Potential for increased resource usage due to object-oriented features for new learners (i.e., redundant codes).

Comparison with C Procedural Design in PPA3:

PPA3 code is more straightforward while writing but less flexible and readable after completing, while project code allows better data encapsulation and abstraction, but sometimes takes more time to create an OOD design and harder to master the OOD skills.

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.

Observation:

The code generally balances between comments and a self-documenting style. Every single function has a comment indicating its usage.

Recommendations for Improvement:

It would be more readable if deleting the comments of instructions provided in the skeleton code after completion.

Refactor comments to focus on the 'why' rather than the 'how', as the code itself should ideally explain the 'how'.

For more complex logic, include inline comments or block comments that explain the reasoning, or the steps involved, aiding in comprehension.

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.

   Proper indentation, white spaces, and newline formatting overall, however, it would provide a better readability if leaving one line between functions and definition of variables in the main code. For example, the initialize function in the main code.

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)

   There is completely no bug captured when experiencing the game, as sample project provided. The project leaves a clear and comfortable size for the game, providing conspicuous instructions and statistics.

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.

   It does not cause any extra memory leak.

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.
   In my first experience with a team software project, it was pretty interesting but also had its tough parts. Working with others was great for sharing ideas and getting feedback right away, which really helped me learn and make the project better. We split up the work by what each person was good at, which made things go faster. But, it was sometimes hard to get everyone's schedules to match up and to agree on some of the design stuff. If I was in a two-person team, I think having someone to think up ideas with and share the work would've made things easier. A second person could bring different views, especially in parts I'm not so good at, making our software better overall.