COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Ahmet Asan	Kevin Yu
Team Members Evaluated	Yazan Qwasmi	Abdallah Aljayousi

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

Part I: OOD Quality

1. **[5 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.

One positive aspect is that the header files and class names are understandable enough to provide a fundamental understanding of each class, such as variable names. However, understanding possible interactions with other objects and classes would not be obvious without having created an extremely similar piece of code previously. There's a significant lack of comments, which would make navigating through this code at first glance extremely difficult. Additionally, some of the existing comments might actually confuse rather than clarify the code. For instance, within the Game Mechanics header file, near the end of the file, there are commented-out methods that should have been deleted before submission.

2. **[5 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.

Once again, if this were our first encounter with this project, deciphering each and every line would consume a significant amount of time due to a severe lack of comments. However, if methods and headers were understood, the code itself would be comprehensible. There are a few variables scattered throughout the code that lack a clear utility, such as the 'ExitFlag' global variable and other unnecessary variables. Nevertheless, excluding these instances, it's relatively easy to grasp the purpose of each line.

3. **[3 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 OOD

- Creates a simple and organized code with all of the classes
- Good for large scale applications such as the snake game.
- Classes can be reused, for example the arraylist class can be used for player snake and the food bucket.

Cons of OOD

- OOD is a harder concept to understand and implement than procedural design.
- OOD is far less efficient than procedural, this is simply because classes are simply more complicated and take up more space than variable and function declaration.

Pros of Procedural Design

- Procedural design is easier to implement for smaller projects such as the iterations we did for PPA.
- Procedural is more efficient than OOD.

Cons of Procedural Design

- Procedural Design does not allow for large groups of people to work on an extremely large project. Objects and classes keep the code organized and succinct.

Part II: Code Quality

1. **[4 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.

As mentioned in part 1, the code lacks a significant number of comments. I would suggest reviewing the code with someone who wasn't involved and encouraging them to add comments wherever they find the code confusing. For instance, without referencing the definition of 'generateFood,' the creation of the 'headPos' object might be puzzling. Adding comments could clarify the necessity of 'headPos' in such instances.

2. **[3 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 the standard indentation rules and is easy to read and follow. There were no significant changes that could be made.

Part III: Quick Functional Evaluation

1. **[6 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 sole major bug was the potential for a food item to spawn within the snake. The root cause of this issue is extremely simple: on line 110, during the check that ensures the generated food item does not appear at the snake's head. The problem lies in the fact that the code solely inspects the head, allowing the possibility of food spawning within the rest of the snake's body. An implementation that could rectify this would involve utilizing the previously created array list of the snake and ensuring that every part of the snake is blocked off from food spawning.

This was the only major error we found within the code.

2. **[4 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 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.

The largest issue we had was the compatibility between Operating Systems, it was a nuisance to have to uncomment and several issue's led us to only work on one machine. Other than that, the option of having someone to brainstorm with made the process much simpler and more fun. We found that we were most successful when working together in person or through a call, initially we worked independently but found our code often didn't work out.