COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Yazan Qwasmi Abdallah AlJayousi
Team Members Evaluated	Ahmet Asan Keven Yu

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.

   The header files all have a comment explaining their use in the program making it easy to understand what their purpose is, but the number of comments in the code itself is a bit lacking, making it a bit difficult to interpret how each object interacts with each other by looking at the code alone.

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 Features:

- Good utilization of OOD in regards to use different classes to insure robustness of of program. The code is well-structured and follows a clear sequence of operations: getting input, running game logic, drawing the screen, and delaying the loop.
- The use of object-oriented programming makes the code modular and easier to understand. Each class has its own responsibilities, such as Player handling player-related logic and Food handling food-related logic. The use of methods like getExitFlagStatus, setInput, updatePlayerDir etc provides a clear understanding of what each line of code is doing.

Negative:

- Run logic seems to be over loaded with many different functions and features that could cause some readability issues and could have been better organized into a few different smaller functions to maintain cleanliness and maintainability of code.
- The code might lack comments explaining the logic and the role of each function, which makes it harder for others to understand the code and function of each function this was although this was not as impactful as there where good function names.

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) Approach:

Pros:

1. Inheritance: Facilitates code reusability and extension by allowing new classes to inherit properties from existing ones.

2. Polymorphism: Enables objects to be treated as instances of their parent class, simplifying code management and extension.

3. Modularity: Object-oriented design promotes modular structure, making the code more organized and manageable.

4. Maintenance and Scalability: Easier to maintain and update due to its modular nature. Enhancements can be made with minimal impact on existing code.

Cons:

1. Complexity: OOD can be more complex to understand and implement, especially for those familiar with procedural programming.

2. Performance Overhead: Object-oriented programs can be slower and use more memory due to additional layers of abstraction.

3. Steep Learning Curve: Requires a solid understanding of OOD principles, which can be challenging for beginners.

4. Overhead in Design: Designing a system in OOD requires careful planning and understanding of relationships, which can be time-consuming.

C Procedural Design Approach in PPA3:

Pros:

1. Simplicity: Procedural programming is straightforward and easier to understand for simple tasks.

2. Ease of Learning: Easier to grasp for those new to programming, as it involves a more linear and less abstract approach. Almost everything is on one file/page making it easy to scroll

3. Predictability: The flow of execution is linear and predictable, making it easier to follow and debug.

Cons:

1. Scalability Issues: Not well-suited for large, complex applications. The code can become unmanageable as the project size grows.

2. Lack of Modularity: Difficult to maintain and update, as changes in one part of the program can affect the entire codebase.

3. Code Reusability: Limited support for reusability, leading to potential duplication of code.

4. Difficulty in Managing Large Projects: As projects grow, procedural code can become more difficult to manage due to its linear nature and lack of abstraction.

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 code is generally well-structured and follows a clear flow, which aids in understanding its functionality. However, there are areas where comments could be improved or added to provide more clarity.
Redundant comments: There seems to be a few instances of redundant comments like
Line 59: `// MovePlayer gets input already`
or
`// Wraparound logic` Being written multiple times in movePlayer() function

Code Duplication: The code for incrementing the score is duplicated. This could be replaced with a single function call that takes the increment amount as a parameter. For example:

```
myGM->incrementScore(10);
```

Numbers: The numbers 4 and 5 are used to determine the type of food consumed by the player. These could be replaced with named constants to make the code more readable and maintainable. For example:

```
const int NORMAL_FOOD_INDEX = 4;

const int SPECIAL_FOOD_INDEX = 5;
```

Unused Function Call: The function line106: myGM->getExitFlagStatus() is called, but its return value is not used. If this function is meant to check if the exit key was pressed, it should probably be used in an if statement.

All this is to say that was not detrimental to the readability but would help follow best development practices

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.

   There weren't any noticeable shortcomings in whitespace or indentation as code was fairly well organized. the code generally follows good indentation, uses sensible white spaces, and deploys newline formatting for better readability. Each function and logical block within the functions is clearly separated by newlines. The indentation is consistent, which makes the code easy to read and understand.

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)

   Although the game worked properly the entire way, its main issue was how buggy it was while playing. the board would appear almost as if its glitching on and off every few seconds while playing. This is a purely cosmetic issue and not one to do with the playthrough of the game itself, but it causes the game to feel laggy and not generally appealing to the eyes while playing. a proposition for the root cause of this issue could be an incorrect or poor use of the clear screen function, causing the game to appear this way. a possible fix for this would be a proper implementation of the clearscreen. this issue could have been caused from my laptop itself being buggy but as this has not happened before i am assuming it is an error with the code itself.

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 memory leakage was observed.

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.

   Developing a C++ snake game for our first joint software project was both an enjoyable and difficult experience. Communication and division of tasks worked well as we outlined the game mechanics and divided coding responsibilities. Different coding styles occasionally caused us issues when coordinating code integration. In order to simplify communication and have a more smooth integration, we discovered how important frequent code reviews are. The innumerable discord calls we hopped on before and during the project also helped strengthen our teamwork skills and taught us how to go about planning a project like this beforehand, and work on it moving forward. All things considered, it was an excellent learning experience that improved our programming and teamwork abilities.