**COMPENG 2SH4 Project – Peer Evaluation [25 Marks]**

Your Team Members:

Our Team Name: drmemory leaked;

Mark Atalla - atallm7 & Fady Hanna - hannaf4

Team Members Evaluated Team name: Gattouz

|  |  |
| --- | --- |
| **bali-fall2024** | **bali@mcmaster.ca** |
| **sayedm8** | **sayedm8@mcmaster.ca** |

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **25 marks**. Do not exceed 2 paragraphs per question.

**Peer Code Review: OOD Quality**

1. **[3 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.

The main program loop demonstrates good object-oriented programming (OOP) design principles. The Good object-oriented programming (OOP) design ideas are demonstrated in the main program loop. The links between components are well-structured, and the interactions between things are typically obvious. Modularity is improved by the apparent encapsulation of each object's duties.   
Positive feedback:

• It is clear that the interests of the various objects are logically separated.

• Appropriate application of object-oriented principles such as abstraction and encapsulation.   
Negative feedback:

• More precise annotations describing how certain items interact during the game loop will improve the code.   
• While some function names are self-explanatory, clarity might be increased by include inline remarks on the goals and results.

1. **[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 C++ OOD Approach**:

* **Summarization**: reduces global variables by nudging data and behaviour
* **Flexibility**: makes it simpler to test and troubleshoot objects
* **Recyclable**: Classes and methods can be reused more effectively thanks to OOD

**Cons of C++ OOD Approach**:

* **Harder**: Can be harder to understand for beginners compared to procedural design.
* **Complex**: Dynamic memory management and polymorphism can introduce runtime overhead.

**C Procedural Design in PPA3**:

* **Pro features**: Simpler logic flow, easier to trace individual operations.
* **Con features**: Hard to scale or modify without affecting unrelated components.

**Peer Code Review: Code Quality**

1. **[3 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 provides a mix of self-documenting variable names and inline comments. However, some areas lack sufficient context for complex operations.

**Suggestions to Improve:**

* Use comments to document class and function roles.
* Include explanations for non-obvious logic in loops and conditionals.

1. **[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 generally adheres to good formatting standards. Indentation is consistent, and white spaces improve readability.

**Suggestions to Improve**:

* Avoid excessive nesting by refactoring long functions into smaller ones.
* Introduce line breaks between different logical sections to improve flow comprehension.

**Peer Code Review: Quick Functional Evaluation**

1. **[3 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 technical user feedback)

The game runs smoothly, with responsive controls and clear visual feedback. However, a minor issue was observed with object collision detection.

**The Root Cause might be**:

* The bug may arise from a rounding error or misaligned object boundaries.

**Some debugging recommendations**:

* Use boundary assertions in debug mode to validate collision zones.
* Log object positions to verify state transitions during gameplay.

1. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.

No immediate signs of memory leaks during gameplay. However, a thorough memory profiling using tools like valgrind or AddressSanitizer could confirm this.

**The might be root cause**:

* Improper deallocation of dynamic objects in destructors.
* Lack of delete calls for objects allocated with new.

**Recommendations**:

* Ensure all dynamically allocated resources have corresponding destructors.
* Use smart pointers (e.g., std::unique\_ptr, std::shared\_ptr) to manage memory automatically.

**Project Reflection**

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team’s implementation in addition to your own, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

The compound design with an additional Pos struct is not sensible for this project in our opinion. We think this because the player object deals with the player’s wrap-around and movement while objPos takes and sets the player's position. Thus, it is redundant and makes it difficult to flow through the code because of the many classes.

1. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you’d improve the object design. You are expected to facilitate the discussion with UML diagram(s). Cant do this one tho

Combining these two objects is easier to read and useful as they are both focused on the player. This would make it easier to go through the class since it is one class focusing on the player. However, if an outsider does not know much about the project, they would be worst suited to follow different classes to understand the point. In our idea though, there is a player class so it would be easier to navigate. This idea is shown by the UML diagram below:

