COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Aryan(karnatia) Arnav(goela26)

Team Members Evaluated Chloe Paula

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 logic in the program loop is that food gets printed at initialization, input is collected, the character is checked, which will either cause the player to move, or the program to end. When the player is moved, the code checks if it overlaps with food, or the player's own body. If it overlaps with the food the type of food is checked, and the condition of the food is applied. If the snake self-collided, the lose condition is true, else the snake will stay the same length and just move position. The grid is then printed out using for loops and a global variable, in this case, '#'. Another loop is inputted to check the if the position matches the snake position or food pos, in which case the respective symbol is printed. If the position doesn't match the food or the player, then a space is printed out. The score is gotten and printed. If the win, lose, or exit conditions are met, the screen is cleared and a specific message print based on the condition, ending the program. The program demonstrates a modular design while accounting for concerns for input handling, logic processing, and cleanup. It effectively uses object-oriented principles, within Player and GameMechs classes.

[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 C++ OOD:

- Classes like Player and GameMechs group related data and methods, improving modularity and reducing redundancy
- Objects can be reused or extended without modifying existing code
- Code structure is cleaner, with a clear separation of concerns like logic, input, and cleanup
- Easier to add features or modify components without affecting the entire program
- Dynamically allocated objects are explicitly cleaned up, preventing leaks

Cons C++ OOD:

- Object-oriented structure can be harder to implement compared to procedural design
- Relationships between objects may not be easily perceived, requiring familiarity with class implementations

- Designing object-oriented programs involves more planning compared to procedural programming
- Projects with simple requirements may become unnecessarily complex if object-oriented principles are overused
- Bugs can be harder to trace due to encapsulation and abstraction, requiring programmers to understand interactions between multiple objects

Pros C Procedural:

- The structure is straightforward, making it easier to write and understand for small projects or beginners
- Procedural programs often have less runtime overhead because they avoid the abstraction layers
- Functions directly manipulate data, providing a clear and predictable flow of control
- Bugs can often be isolated to specific functions, making them easier to identify and fix
- Procedural design is well-suited for small, straightforward applications where modularity and scalability are less critical

Cons C Procedural:

- Functions often operate directly on shared data, which can lead to difficulties in isolating parts of the code
- As projects grow in size, procedural code becomes harder to manage
- Heavy reliance on global variables can lead to unintended side effects and makes debugging harder
- Functions and code blocks are typically tied to specific implementations, making them less reusable in other contexts
- Without encapsulation, it is more difficult to locate and change functionality without inadvertently affecting other parts of the code

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 offers sufficient comments which are detailed and explain the purpose of each function. Each data structure has been commented at the top to give the general purpose. One improvement that could be made is to comment on specific lines of code in the program to ensure the reader knows how the program works. All functions, objects, and other variables have also been named in a straightforward and easy-to-understand way. There is plenty of self-documenting code to help the reader understand code functionality.

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 does follow good indentation and there is a good number of white spaces that separate data structures. The spaces that were created make the code easy to read and understand without any confusion from squished lines.

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 snake game offers a smooth, bug free playing experiences. The UI guides the user on the instructions and also implemented a win condition to make it a fully playable game. The additions of numbers which decrease length and uppercase letters which give 10 points are a fun addition to the game.

2. **[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.

The team evaluated did not have any memory leak in their code. The destructor methods were correctly implemented as well as the deallocation of memory of the pointers in the project.cpp file

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?

I do believe the compound object design of objPos class is sensible because it allows you too automatically have the attributes of the Pos struct. When we first created the food and the snake itself the Pos struct in combination with the objPos class allowed to always access the x position, y position, and the symbol. Using a struct pointer it is an easy easy to access the x and y coordinates of the object you are using.

2. **[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).

ObjPos
+ int x + int y + char symbol
+ objPos() + objPos(int xPos, int yPos, char sym + ~objPos() + objPos(const objPos &p) +objPos& operator=(const objPos &m) +void setObjPos(objPos o) +void setObjPos(int xPos, int yPos, char sym) +objPos getObjPos() const + char getSymbol() const

+char getSymbol() const +char getSymbolPosEqual(const objPos* refPos) const +bool isPosEqual(const objPos* refPos) const

The ObjPos class was implemented well therefore no major changes were made in the UML diagram proposed. The proposed improvement is breaking down the struct into two variables x and y. A struct takes more memory, therefore the proposed solution would reduce the memory used in the code and would increase the code readability

Example:

Former: objPos.Pos->y

Proposed: objPos.x