COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members ___chungc47_______ __medinadp_______

Team Members Evaluated ____goela26 __________ ____karnatia_________

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 evaluated team demonstrated a clear interaction between objects of the Player and GameMechs class. The Player class interacts with the GameMechs class to receive input, player positions and to generate food. On the other hand, the GameMechs class ensures the score is up to date and generates food positions, while the Player class manages the player movement and determines a collision between the player and the food. The team effectively used encapsulation within the main program loop, which makes the program easy to follow and scalable.

Some room for improvement in terms of program logic could involve separating certain functionality in the Player class. For example, food collision detection is completely implemented within the movePlayer method, but this can be made into another method of Player class. Similarly with self collision detection and increasing the player's length, which is entirely handled by movePlayer.

[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:

- Better for complex and scalable projects, while C procedural is better for smaller projects since it difficult to scale projects, like the snake game, since the functions are generally dependant on one another or hardcoded, making modifications more complex and limited.
- The use of different classes, such as Player and GameMechs, divides the program into separate components that focus on a different aspect of the program's functionality. Each class is easier to debug and modify without the need for modifying other classes.
- Encapsulation is also more complex as it requires more effort to implement in the C procedural approach. Encapsulation is naturally integrated in C++ ODD approach due to the compiler when private and public controls are used in the class, which hide members or implementation from the rest of the program. This prevents modifying data by accident, which reduces the risk of errors.
- While C procedural approach makes it difficult to scale projects since functions are generally dependant on one another or hardcoded and limited, C++ OOD allows to scale projects, such as adding more players

in the snake game. Additional features are integrated into the program more clearly than procedural programming.
- Improved readability due to the clear relationship between each class and the main program.

Cons of C++ ODD Approach:

- Unlike C procedural approach, which is easier for beginner programmers to understand, C++ ODD can be a learning curve since it requires understanding how classes and objects interact, as well as maintaining the rule of six during the initial design.
- In the snake project, the use of dynamic memory allocation increased the risk for performance issues due to memory leakage.

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.

    - There are comments describing the general function of the code (e.g. What happens when food is eaten) but there are not as many comments describing how the function is implemented
    - I would improve the code by removing irrelevant comments (e.g. Notes made during code development) and by trying to add more details describing how the functions are implemented

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 has good white spaces to separate the code, and it does deploy newline formatting. The indentations in the if statements, loops and switch statements are all clear and makes it easy to tell what is included within each loop/statement.
    - A way I would improve the indentation is by making sure the brackets in nested if/switch statements/loops are clear

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 experience. The game processes smooth user input, allowing timely responsive direction changes and forced exit, with proper losing or forced exit messages. Also, the game also maintains the game board visible even after the game ended, which adds to the user experience.

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 Snake Game does not cause memory leak.

Project Reflection

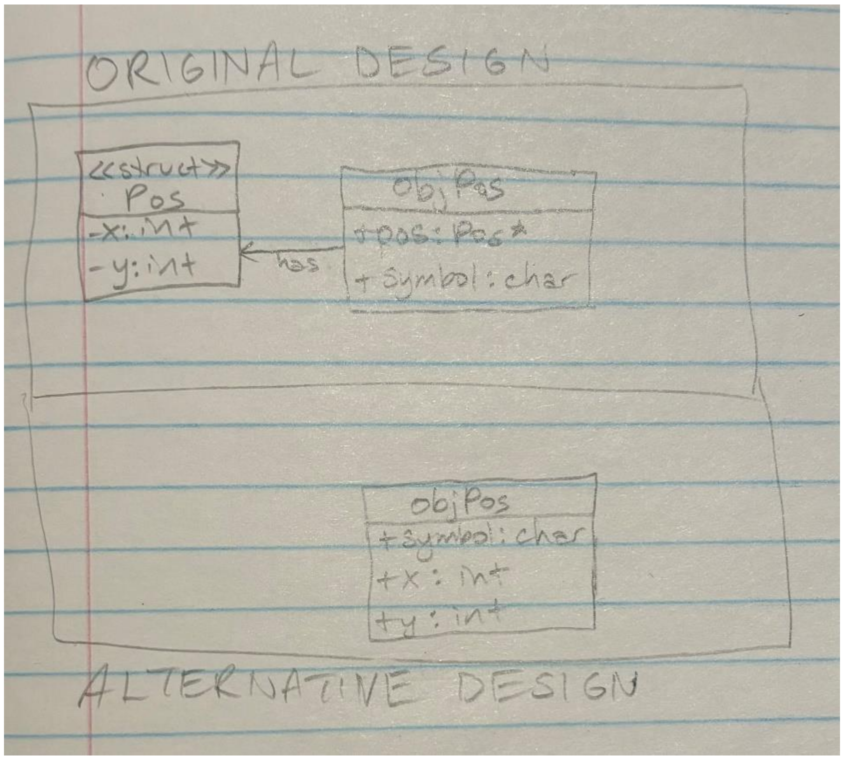
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

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

   It is somewhat sensible to separate the objPos into two parts: its position and its symbol. On one hand, it is a good OOD decision and makes the position separate from the symbol so that if further operations need to be done on the position there is a place for that. On the other hand, since objPos is a simple class with only 3 variables the extra struct, Pos could overcomplicate the code implementation.

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. <u>You are expected to facilitate the discussion with UML diagram(s).</u>

   For this project specifically where there are no extra operations done on the position, we would remove the Pos struct and implement the x-position and y-position separately in the objPos. Doing this, the alternative objPos class would contain x, y and symbol shown in the UML diagram.


ORIGINAL DESIGN
<<struct>>
Pos
-x: int
-y: int
has
objPos
+pos: Pos*
+symbol: char
objPos
+symbol: char
+x : int
+y : int
ALTERNATIVE DESIGN