COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Himesh Mistry and Ivan Nazaire

Team Members Evaluated const students team[2]

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.

In the main program loop, we can interpret how the objects interact with each other, due to the clean code (everything is spaced out and indented in an organized manner for readability), the variable and function names are self-explanatory and, comments to understand what the block of code is doing. We can see that in the Initialization, they initialized the necessary objects. In GetInput, the user input is read through the gameMechs class and the getInput function. In RunLogic, the user ends the game or moves the snake based on the inputs. In DrawScreen, the board, player and food are drawn. The LoopDelay controls the delay for the game speed. In CleanUp, the dynamically allocated objects are deleted for memory leakage.

Some positive features were that they followed the rule of minimum 4. They have a default constructor, copy constructor, destructor and copy assignment operator. The classes they created have a purpose. Player handles the snake, GameMechs manages the score, input, etc and Food generates the food. They also used Encapsulation. For Player and GameMechs class, they have private and public variables/functions to expose only relevant features. They also used Dynamic memory, (new and delete). This creates flexible and scalable objects. A negative, is that they could have removed their debugging in the main to present the code more cleanly.

2. **[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:

- Able to reuse code (Inheritance)
- Easier to maintain
- Scalable
- Improved code organization and data safety (Encapsulation)

Cons:

- More complex to use/incorporate object-oriented programming.
- Need to follow the rule of minimum 4. Need to have a default constructor, copy constructor, destructor and copy assignment operator.
- Not required for small projects. Useful in bigger projects or scalable projects.

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.

Basic comments are given such as, which function is the constructor, destructor, etc and also on some for loops or line of codes. However, there is an inconsistent use of comments. For instance, when creating the functions for the array list (removeHead, removeTail, etc), they added comments for the head functions, but not a lot for the tail functions.

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 formatting of the program is very good due to the good indentation and sensible white spaces. This makes it easier for a programmer to understand a function and to find a variable, class or function.

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 team's Snake Game does offer a smooth and bug-free playing experience. The movement of the snake reacts well to user inputs, the snake grows fluidly when it eats food, the wraparound looks correct, and the snake dies after its head collides with its own body.

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's Snake Game does not cause a memory leak as all the allocated heap elements are appropriately deleted in the code.

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
     11 unique,   113 total uninitialized access(es)
      0 unique,     0 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      0 unique,     0 total,      0 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
```

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:

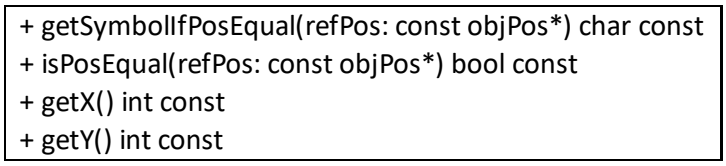
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

We do not think the compound object design of the objPos class is sensible. While we do believe it is useful for when we want to add additional data about the positions without affecting the rest of the objPos class, it is redundant in the context of a simple snake game where the only things in the struct are its x and y positions. It is pretty much equivalent to storing x and y as members of objPos but just makes it a bit more complicated.

By using struct Pos, the dynamic memory allocation for pointing to the struct causes unnecessary heap memory usage. The use of new and delete increases the risk of memory leaks if the implementation of the copy assignment operator and destructor is incorrect.

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



We would improve the object design to follow proper encapsulation practices. If we put x and y in private, we can prevent any accidental modifications to snake's position. The symbol variable is also moved into private. The destructor in objPos can be left empty since there is now no need for dynamic memory allocation. Two getters are added in public to provide controlled access to the private x and y variables. The getter for the symbol and the setter (setObjPos) are already implemented from when the struct Pos was there.

Using these additional functions throughout the rest of the code makes the code function the same as when the struct Pos was there. For example, inside the movePlayer() function in Player.cpp, the x and y coordinates of the temporary head position can be stored in separate variables. Then after the input processing and wraparound logic is done, the new x and y coordinates are set into the temporary head position and then inserted into the actual snake head position.