COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members ____Mahum Khawaja__ ____ Sehaj Kaur _____

Team Members Evaluated Member 1: Paul Stoica MacID: stoicap StudentID: 400514116

Member2 MacID: harria49 Member 2 StudentID: 400502398

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 code is easily interpretable. The team used a separate class for Board and modularized it making the main clean. There are some good features in the code that add to the game and random food characters generate at random positions. Debugging test cases are also shown but commented out. This shows how they tested different key buttons. Each key, 'a', 's', 'w', 'd', is also in its own variable in the Player file. This makes the code very intuitive to understand.

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

- Every object can exist independently from other objects
- Private data members and information cannot be accessed by the public
- As member functions and member fields are customized for the object, the code makes intuitive sense when it interacts with other objects

Cons

- Very, very, complicated. Understanding one line of code can take switching between multiple pages of code. The added complexity of how a constructor is defined, where it is declared and then the pieces that it interacts with, makes implementing code difficult.
- It's difficult to test your object if it depends on other things. And you would have to do unit testing
    - In a procedural approach, all code is additive, and you would test every new portion of code added, thus you can test at every stage of your 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.

There is sufficient commenting, and it is easily readable by the user. The explanation provides enough context in the main and Board class. In the ObjPosArrayList, the copy assignment operator is commented as 'overload', which is incorrect but may be an oversight and does not affect the code.

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 is easily readable and has good indenting and white space. All information is printed on a new line. The comments point to different parts of the code you can refer to if the code logic is unclear, such as the implementation of Board. They commented where to look for more details on Board in their main code. They also wrote in a very clear and clean format.

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)

There are no functional bugs or issues with the game. Well done! There is correct implementation of game board, arrow key movements, inputs, random food generation, collision detection and the snake size grows. I would recommend providing instructions on some game functions and effects such as 'lazy' and 'blind' for clarity but that does not affect the main piece of the code.

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.

There are no memory leaks in the code as per the memory profiling report acquired through dr memory. It shows 0 bytes of memory leak. Upon inspection, destructors were implemented so everything was deleted.

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 advantage of using the objPos class with the Pos struct is that is simplifies the tracking of the x-y coordinates as it modularizes it. This encapsulation ensures that these related data variables are used together and tracked simultaneously, improving maintainability and security of the variables. Since food and the main game character require the same things, it simplifies the code as we can use Pos struct as a template to handle these pieces of data, promoting reusability. This method also allows for flexibly memory handling as Pos can be created,

destroyed and modified. The interaction of objPos and ObjPosArrayList helps grow the snake as insertHead and removeTail is used. However, this also means that the heap allocation added extra complexity and risk of memory leakage. An alternative design would be to have x and y as direct field data members as opposed to being modularized. It would simpler but would sacrifice modularity. The design is sensible given it aligns with OOD principles.

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>

A less memory efficient but alternative design of ObjPos is to declare the x and y coordinates as field data members. They would have their own getters and setters.

<u>objPos</u>
- Xpos: int - Ypos: int + symbol: char
+ objPos() // Default constructor + objPos(xPos: int, yPos: int , sym: char) Constructor with position and symbol + setObjPos(o: objPos) : void + setObjPos(xPos: int, yPos: int , sym: char) : void + setXpos() : void + setYpos() : void + getXpos(): int const // Getter for X-coordinate + getYpos() : int const // Getter for Y-coordinate + getObjPos() : objPos const + getSymbol() : char const // Getter for symbol + isPosEqual (refPos: const objPos*) : bool const + getSymbolIfPosEqual (refPos: const objPos*) : char const