COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Elijah James [jamese13] Grace Jonker [jonkerg]

Team Members Evaluated Kamyar Lakdashti [lakdashk] Kai Hughes [hughek26]

(can't find their names, only their macIDs)

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 structure is easy to understand. That is, the flow of Initialize( ), GetInput( ), RunLogic( ), DrawScreen( ), LoopDelay( ), CleanUp( ), and LoseScreen( ) makes it clear how the program is structured and that it achieves a user interface and display. It is also clear that Player, GameMechs, and Food classes are used to run specific, related, game functions and hold information about specific states, for example storing the exit status. One negative feature is that it is unclear how the three classes interact, and we know that they must interact to accomplish things. For example, the player colliding with food on the gameboard requires interaction between the player and food classes, and the specifics of these interactions aren't visible in the main program loop. The names of the classes make it slightly easier to understand how they interact with eachother like for objPosArrayList, it is clear that it is an array list containing instances of the objPos class. However, for things like GameMechs, there is no obvious relationship between it and other classes just by its name.

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 of OOD / Cons of Procedural Design:

- There are fewer global variables making it more memory efficient.
- In OOD you can reuse names meaning there is lower risk of mistakes or accidental altering of a variable.
- Easier to collaborate on the project since some iterations don't necessarily require you to fully understand previous iterations.
- Adding some new features is easier since there are already templates in the form of classes.

Cons of OOD / Pros of Procedural Design:

- Interactions between classes is unclear in the main program loop.
- It is more complicated to understand object-oriented design.
- More files to keep track of and organize.

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 no comments in the objPosArrayList files and few in the GameMechs files making them more challenging to interpret. In addition, some of the comments are also redundant or no longer relevant. For example, in the "updatePlayerDir( )" function, the lines considering each character have a comment that contains only the character again: "case 'd': // d". This adds nothing to the understanding of the code and would be improved by writing something like "player wants to go downwards". They also left in multiple [TODO] comments even though the tasks have already been completed which is confusing. The project file has slightly more useful comments, but it would be better to have more explaining the specific steps instead of just the objective of a function or loop.

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 follows very good indentation and uses sensible white spaces making it easy to read and understand what sections of code are working on different things. For example, in each header file the methods and members are visually organized into private, public, and then into constructor/destructor, special members, and getters. Functions within each class are also well organized in the same order they are declared in the header file making them easy to locate. In addition, nested loops and switches use indentation to help visualise the different conditionals and their dependent actions.

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 does offer a very smooth and bug-free experience. I did not encounter any buggy features while testing the game and thus I have not proposal on potential debugging approaches.

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?

Yes, the compound object design of the objPos class is sensible in some respects by using a Pos struct to encapsulate the x and y position values of an object into a single private position property when created with the objPos class. This separates the position from the rest of the object's behaviours, such as its symbol, and allows for the dimensions of the space to be altered in the struct without affecting the main functions of the objPos class. For example, if we start using objects in a simulated 3D space, we can add a "int z" to the Pos struct and only need to add the required "pos->z" to the constructors, copiers, getters, setters, and other functions involving position, while simultaneously leaving all the object's other behaviours as is. The use of a struct also guarantees that if an object has a x-coordinate value, it must also have a y-coordinate, regardless of if any other properties are defined.

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).

What if the private properties of the objPos class were classes themselves? The UML diagrams would be:

Position
- x : int - y : int
+ Position() + Position(xPos : int, yPos : int) + setPosition(xPos : int, yPos : int) + getX() : int + getY() : int

Symbol
- value : char
+ Symbol() + Symbol(sym : char) + setSymbol(sym : char) + getSymbol() : char

objPos
- position : Position - symbol : Symbol
+ objPos() + objPos(x : int, y : int, sym : char) + objPos(object : objPos& const) + operator=(object : objPos& const) : objPos + ~objPos() + set ObjPos(x : int, y : int, sym : char) + getPosition() : Position + getSymbol() : Symbol + isPosEqual(refPos : objPos* const) : bool + getSymbolIfPosEqual(refPos : objPos* const) : char

Pros of this "counterintuitive" design:

- Improved Encapsulation where the Position class handles all coordinated related functions, and the Symbol class handles all symbol related functions.

- Flexibility for extending the Position and Symbol classes without affecting the other classes.
- Simpler maintenance and unit testing since each class is smaller and more specified in its uses.
- The separation allows for improved readability on the purpose of each class, as well as better reusability for functionality outside of the objPos class.

Cons of this “counterintuitive” design:

- Having three separate class increases complexity for simple cases and can feel like overengineering.
- Since values are being passed between the different classes, there is an increase change of the values being accidentally changed or lost in functions.
- Storing separate objects in their own classes will require more memory usage and is inefficient.