COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members ______pathaa24_________ dougla15

Team Members Evaluated ___nazairr____________ _mistrh13____________

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.

- ➔ Yes, it is easy to interpret the interactions between the various objects within the program logic. We identify each method being called and its usage within the code as well as which class it is obtained from.
- ➔ Some positive features that we observed from the team's program logic are:
  - ◆ The use of functions are appropriate and are relevant to the main logic portion
  - ◆ The code is not overly complicated and can be understood on how the logic is carried out
- ➔ Some negative features observed:
  - ◆ Within runlogic(), the methods such as updateplayerdir() and getexitflagstatus() could be called within moveplayer(), so code is more object oriented and modularized.

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:

   - ➔ Easier to work as a team because each person can work on a different class and then incorporate them into the main program
   - ➔ OOD approach allows each method to have its own specific task/block of code, so when an action is required, it is more efficient to just call the method from the class
   - ➔ Main program loop is much more readable because each function is modularized into different class files

   Cons:

   - ➔ Harder to debug because of the increased complexity of interactions between classes
   - ➔ Pos struct was a bit redundant, a lot of methods needed to be called just to get the position of an object (ex. player -> objPosArrayList->objPos->pos->x)

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 sufficient comments that specify the role of each code block making the it easy to understand
- ➔ The variable and function names aptly describe what the code is doing so comments aren't always needed.

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.

- ➔ More whitespaces would have made the code more easily readable and helped to separate different sections of the code. This is especially noticeable when there are lots of nested loops and if statements like in the drawScreen() function.
- ➔ The indentation is good and is consistent throughout.
- ➔ The print statements use newline formatting for good readability and printing the controls and game mechanics for the player is a good idea.

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 buggy features. The snake game runs smooth and bug-free including the special food feature. However, for future projects it is best practice to include copy constructors and the copy assignment operator (deep copy) in all of your classes.

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.

- ➔ No, there are no memory leaks.

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 don't think the compound object design is sensible.
   - There's no reason why the x and y values couldn't be stored in objPos because the symbol is stored there too
   - The programmer has to call a lot of methods just to get the x and y values (ex. player -> objPosArrayList->objPos->pos->x) which takes a while to type out and makes the code less readable

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 better way to implement the objPos class would be to have the member variables int x, int y, and char symbol instead of having a struct Pos which has int x and int y. Additionally, all 3 of the variables would be private instead of public to make the class more secure and prevent their values from being unintentionally changed. We would also have to create new member functions int getXPos() and int getYPos() to return the x and y int values.

UML Diagram of proposed changes:

objPos
- x : int - y: int - symbol: char
+ objPos() + objPos(xPos: int, yPos: int, sym: char) + setObjPos (o: objPos): void + setobjPos(xPos: int, yPos: int, sym: char): void + getObjPos(): objPos const + getSymbol(): char const + getXPos(): int const + getYPos(): int const + isPosEqual(refPos: const objPos*): bool const + getSymbolIfPosEqual(refPos: const objPos*): char const