**COMPENG 2SH4 Project – Peer Evaluation**

Your Team Members Audrey Gamboa and Alex Walmsley

Team Members Evaluated Rebeca Gaston and Juvani Kurunabalan

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

**Part I: OOD Quality**

1. **[6 marks]** OOD is about sensible code modularization. Looking at the header files of each object, can you easily interpret the possible behaviours of the objects involved in the program, and how they would interact with each other in the program? Comment on what you have observed, both positive and negative features.
2. **[6 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.
3. **[5 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.

**Part II: Code Quality**

1. **[5 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.
2. **[4 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.

**Part III: Quick Functional Evaluation**

1. **[8 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 a technical user feedback)
2. **[6 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause of the memory leakage.

**Part IV: Your Own Collaboration Experience (Ungraded)**

* + - 1. Tell us about your experience in your first collaborated software development through this project – what was working and what wasn’t. If you are a one-person team, tell us what you think may work better if you had a second collaborator working with you.

**OOD Quality**

1. By observing the header files of each class, I can easily understand the behaviors of each object and their intended use in the game. Each class has members functions that are grouped into getters, setters, and available functions which helped me to identify what each object can do. Additionally, at the top of each header file shows what other classes are needed for its functions which further help me to understand how the classes interact. For instance, the Player class requires a reference from the GameMechs class and also includes the objPos class which tells me that each instance of a Player is an objPos, meaning it has an x,y coordinate and a symbol, as well as requires the game board information from GameMechs to move or update. Next, the arguments passed into each function have sensible names that also aid in interpreting the behaviours of the objects. An implementation of a food class rather than including ‘food’ features into the GameMechs class was a smart approach as it keeps the classes and their interactions simple.

2. The main program is written very efficiently with effective use of the classes. Since the classes are well implemented, there was little coding required in the main program. I can observed that after instantiating an object of each class in the Initialize() function, all the required actions in the game are already taken care of. Also, the order which the objects were instantiated and used shows how the object interact. For example, I noticed that the Player object needs to be instantiated and its position retrieved to generate food. The only required action left is to update and move the player after checking the user input, and all else will follow. I noticed that instantiations of an objPos object for temporary use was declared but never used (line 73) and the getPlayerPos() was called but wasn’t returned to any variable (line 74), proving redundant. Additionally, I noticed there is some code in the main program that could have been implemented in the respective class, improving the main program logic and emphasizing the OOD approach. The RunLogic() function of the main code checks the input for the exit key as to ensure the proper exit procedures can follow, however this section of code can be added to getInput() of the GameMechs class, which would leave the main program code only for object interactions.

3. pros and cons of OOD approach

Pros:

-improves the readability of the entire program but especially in the main code

-global variables are protected against mistakes – they can never be altered in the main code

-no more need to keep track of several different random variables

-it’s easier to implement any ‘actions’ in the game since you do not need to follow a certain sequence to get the action done, you only need for the objects to interact

-much more flexible

Cons:

-difficult for beginner coders to code with this approach since it can be challenging to manage the tabs of different classes and understand how the classes interact

-if many classes are interacting with each other, it can be difficult to debug as you may not know which class or classes the bugs are coming from

**Code Quality**

1. The comments throughout the code are very well detailed and easy to understand. When going through each method, they were all well explained and even took me through the process of how the code works. They also did not add any redundant comments within their code, making it less overwhelming for the reader to follow along. For example, they avoided adding comments like “finds the score value and returns it” to the method getscore(), as the person reading can easily figure out what the method does by reading the name.

2. Throughout the entire code, it is properly indented and spaced out. The style of formatting is consistent with the way we were taught in class and is very easy to read. They never deviate from this formatting once throughout the entire code, making it very easy to follow through and read the code.

**Quick Functional Evaluation**

1. The game runs as expected, and I did not notice any bugs within it. The foods add the right amount of body, the snake wraps around, and the game properly ends when you hit yourself. The bonus foods implemented were a neat addition that can speed up the progress within the game.

2. The game does not have a memory leak, and all the allocated memory has been freed up properly.