COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members

Team_Masters: Vince Amado (amadov) and David Elyahky (elyahkyd)

Team Members Evaluated

Taj and James: Taejin Eom (eomt) and James Shakespeare (shakesj)

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.

<u>Peer Code Review: OOD Quality</u>

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 different classes are easy to understand and were implemented in a neatly manner, following the recommended workflow in the project manual. The member functions in the Player.cpp file is constructed correctly but there are some redundancies. For example, under updatePlayerDir(), the default case of the input direction is repetitive as the switch cases are already handling these conditions. This redundancy increases the complexity of the code without providing additional functionality. Overall, the project shows good structure and adequate memory management, but it could improve upon the organization of the code.

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 C++ OOD / Cons of Procedural Design:

- Code is broken down into reusable objects and classes which allows for easy modularity to extend distinct blocks of code.
- Procedural design is lengthier because it only consists of one file compared to OOD.
- Memory management is more efficient since there are fewer global variables.

Cons of C++ OOD / Pros of Procedural Design:

- OOD is harder to track since the user needs to go from file to file to follow the objects such as which file or class they are located.
- Procedural code is more straightforward for simple tasks because it focuses more on different features for each function.
- OOD is more complex because it requires understanding of concepts like inheritance and class interactions.

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.

The code does include some sufficient comments that made it easier to review and understand the code. The comments provided a lot of details to what each notable line does and how it connects to the different classes especially the Player.cpp, GameMechs.cpp, and Project.cpp. However, in the objPos.cpp and objPosArrayList.cpp, there are some functions with no comments which are inconsistent compared to other functions that have comments for each line of code. To improve this, we would suggest adding descriptive comments to the functions in these files, particularly complex operations involving pointers and remove the ones that are for instructional purposes like "more actions to be included."

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 generally follows food implementation, sensible white spaces, and newlines to improve the code's readability. Each of the member functions are properly indented and not compressed to each other. However, there are a few instances in which the spacing could be improved upon. For example, in Player.cpp, under updatePlayerDir() and movePlayer(), the use of curly brackets could be spaced out in which the open bracket is only present in one line and the closing bracket is also isolated in its own line to visually separate different sections of logic. If the spacing Is graded more strictly, we can also point out the inconsistent spacing when using the arrow operator when accessing objects. For example, some lines have a space in between the arrows like if(snakeheadCopy.pos -> x > 28) and some are compact if(snakehead.pos->y == 13).

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 smooth, bug-free playing experience with a clear movement of the snake whenever it consumes a food item. The collision whenever the snake intercepts with its body is also correctly implemented that terminates the program, causing the player to lose. However, the exit key which was escape was not specified in one of the print statements in the Project.cpp file. This could benefit the player when to end the game whenever they want and also for users to know which key to press to check for memory leaks.

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 project does not cause any memory leaks after running DrMemory which means that the heap allocation and deallocation are handled successfully.

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 compound object design of the objPos class is not very sensible. This unusual implementation is quite redundant and serves to create confusion when trying to access the different data members within the objPos class. Throughout the project whenever trying to access these data members through getter methods, the programmer would always have to add extra specifications after the getter method is called. For example when trying to access the X position of the head element of the player list one would need to write: playerPosList->getHeadElement().getObjPos()**.pos->x** , when commonly the bolded section would not need to be stated.

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).

A possible alternative objPos class design would be to include X and Y as data members directly in the objPos class, rather than including a struct with these data members then passing a pointer to it. This would then necessitate the addition of more getter methods to access the data members. A possible UML diagram would be:

- xPos : int - yPos : int - symbol : char
+ getPosX() : int const + getPosY() : int const + getSym() : char const

Note: the above UML diagram does not include all methods of the class, only the ones that would be modified.

Despite having more methods, this implementation should allow for code to be much simpler. Going back to the example mentioned in Q.1 , in order to access the X position of the head element of the player list one would need only write: playerPosList->getHeadElement().getPosX() , a big simplification when compared to the original: playerPosList->getHeadElement().getObjPos()**.pos->x** .