**COMPENG 2SH4 Project – Peer Evaluation [25 Marks]**

Your Team Members \_\_\_Akinniyi Chidiebube\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Team Members Evaluated \_\_\_Aegir Wanga\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Joshua Xue\_\_\_\_\_\_\_\_\_

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 consists of just class specific instructions, making it easy to understand to process. |

1. **[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.

|  |
| --- |
| The OOD approach allows one to perform more actions more easily than in the PPA3 approach.  Take for example the player list, to get the size of the player and constantly update it, requires a loop and repetition, in OOD a function can easily by attributed to the player and the function called at each instance.  When going through the code, it might get confusing to understand which class each element comes from and its properties. |

**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 naming of the variables is abbreviated, unless one identifies the initialisation of the variable, the code doesn’t make sense, e.g. n and m in the move Player function, t, ty and tx in the game mechanics generate food. I would prefer them to be written with easy-to-understand names like row (m), column(n), target (t), flag(f).  Comments were provided during declaration and implementation which allowed for better understanding of their code, considering the variable names, the comments could have been a bit more, to better understand the process within a function as well. |

1. **[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.

|  |
| --- |
| Yes, it has proper indentation, it could use more white line spaces for better readability but its not so bad that its an actual issue. |

**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)

|  |
| --- |
| Program runs smoothly without bugs. |

1. **[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 bytes of 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?
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).

|  |  |  |  |
| --- | --- | --- | --- |
| 1. I think the addition of the Pos struct within the objPos was sensible. The members, x and y, within the pos are public members, the Pos structure adds a bit of protection to them as one cannot accidentally change these values during code as easily as if x and y were just typical member functions. 2. Another way, it could be done would be to simply declare integer members x and y or integer array pointer pos (that would contain a 2-element list x and y) on the private scope, then use setters to change specific values and getters to get specific values.  |  | | --- | | objPos | | - x: int  - y: int  -symbol: char | | + objPos()  + objPos(xPos: int, yPos: int, symbol: char)  + setObjPos(Obj: objPos): void  + setObjPos(xPos: int, yPos: int, sym: char): void  +setposx(): void  +setposy(): void  + getObjPos(): objPos const  + getSymbol(): char const  + getSymbolIfPosEqual(refPos: objPos\* const): char const  +getposx(): int  +getposy(): int  + isPosEqual(const objPos\* refPos) : char const  + ~ObjPos() |   UML Diagram |