COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members _____Margaret Bui_____ ___Zahra Kazmi_____

Team Members Evaluated __Matteus Jedryk_____ ___Denha Marven_____

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.

   It is straightforward and easy to interpret how the objects interact with each other in the program logic. There are several interactions between GameMechs and Player, one of which myPlayer holds reference to myGM to retrieve input and modify the score and player state. There's also interaction between objects of Player and objPosArrayList/objPos, GameMechs and objPos, etc. This code allows for good encapsulation and modularity in which each object has a specific responsibility and dynamic interactions. On the other hand, this can lead to a situation of higher dependency in which a change in (for example) GameMechs could force changes upon the Player class. This makes for reduced flexibility when coding and can be annoying when making modifications and testing.

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	Cons
C++	• Separate files based on the class function for a smoother design process, i.e., more organized, shorter codes, information hiding between private and public objects • C++ supports encapsulation and polymorphism so that a base class can be used as a template for related classes to share certain behaviours	• Pass by pointer mode can involve complicated referencing/dereferencing setups • Addresses in the pointer can be changed without the const qualifier in Pass by Pointer

	• Less stack memory is consumed in Pass by Reference	
C	• All functions/classes are placed into a single file they can be called directly rather than initializing new pointers to each one • One main file with all functions/classes can make coding faster and more efficient	• All functions/classes are placed into a single file, making for lengthy codes, so it is harder to read and navigate • C does not support encapsulation nor polymorphism

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.

   Each file has sufficient commenting for each block of code, effectively explaining the purpose and use for future references. For example, in the objPosArrayList.cpp, each function has a concise explanation of what is happening in the list and subsequently the snake. This team chose to incorporate the food generating mechanism in the GameMechs class, keeping the Snake Food as an objPos member (of GameMechs class), whereas our team created a Snake Food class. Their method is well commented and called. Overall, their code is easy to understand, even for someone who may not be familiar with the project manual. For improvement, they could delete the comments that were provided as guidelines.

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 good indentation and formatting for better readability. They indent when implementing functions with curly brackets {}, like in for loops, if statements or classes. The user interface is also straightforward in its design, offering instructions on how to play the game, the score and lose/exit code messages. Also, long lines of code, such as the declaration of headPos{}, were put onto new lines to avoid going off-screen and improve readability.

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 indeed offers a smooth and bug-free playing experience. Both the self-collision and backspace to exit are working, and there are no errors or delays with the wraparound or increasing length of the snake.

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 caused no memory leak. After running a drmemory report, there were 0 bytes of leaks and 0 bytes of possible leaks to be found. This is supported by the fact that this group properly deallocated memory on heap in the cleanup() function, and in every other class file where memory was stored on the heap, they added a destructor to delete all pointers declared in the function(s).

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 objPos class has a pointer to Pos struct, which holds x and y coordinates, and a symbol representing the snake body and food. The Pos struct essentially just contains int x and y, representing coordinates. This compound object design seems sensible because it supports encapsulation and modularity for OOD; Pos struct encapsulating the coordinates allows for the objPos class to not take on that responsibility, instead managing storing and handling the symbols. Furthermore, this flexibility allows for adjusting either objPos class or Pos struct with no disruption to the other or program. However, compound design adds a layer of complexity for a relatively simple use, and possible confusion. It might also make for better readability to store the coordinates directly in objPos class.

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).

Alternative objPos UML diagram

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

+ isPosEqual(refPos : const objPos*) : bool const + getSymbolIfPosEqual(refPos: const objPos*) : char const

The UML diagram seen above declares the position (x and y) and character symbol as public data members, eliminating the Pos struct. This design is counterintuitive because it is redundant. The Pos struct was intended to be a reusable representation of the object data, but embedding this in the public access specifier could make the code harder to maintain or add new features. For example, getting and setting the objPos position could change the value of these variables every objPos instance, which is inconvenient when there are multiple objects (snake head, snake tail, or food) that use this class. Additionally, this design lacks encapsulation – a benefit of C++ which hides internal implementation from any external interactions. As public data members, there is less distinction between the properties of the object, especially when working with larger systems that may involve more complex attributes. Encapsulation should provide information privacy, clarity and maintainability, but this alternative design does not.