COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Christina Bridges (bridgc1), Sonia Parekh (pareks5)

Team Members Evaluated Elizabeth Mark (marke3), Sapna Suthar (suthas4)

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.

Declared variables in the main program loop exhibited intuitive variable names (i.e. myPlayer, myGM) of correct pointer type as well as method names. For instance, the variable head of objPos type was assigned with the r-value of the player position list called getPlayerPos() and only the head element data was accessed via the arrow operator to getHeadelement(). Initializations were completed correctly with proper heap memory allocation and organized efficiently depending on the global pointer used. The single food object was also initialized and the position of myPlayer passed in as expected, as the generateFood function should ensure the food position differs from the player position. Note that myArrayList of objPosArrayList pointer type is initialized, allocated memory, and deleted from the heap, however it is never used. The creation of this pointer wastes heap memory and decreases the efficiency of the program, although almost negligibly. The iostream library was also included at the top of Project.cpp although it is not necessary to include for the functionality of the program despite being included in the template.

No procedural decomposition was observed, as data from initialized objects were accessed using getters to avoid compromising class private variables. Thus, proper encapsulation techniques were used and reusability achieved. Member functions were properly invoked using '->' for pointers and the dot operator to access variables of objects in Project.cpp. In the DrawScreen() function, thisSeg was initialized to data type objPos with the r-value playerPos->getElement(k) to iterate through all snake segment positions to check for equivalence to the current x and y positions on the game board (represented using i and j variables). The .cpp and .h files for the initialized variables in the main run logic of Project.cpp include required constructors and destructors, with the objPosArrayList files satisfying the rule of the minimum 4 member functions. All input parameters and return data types are correct with intuitive and concise code while including necessary header files. This ensured an organized function setup for ease when interpreting multiple function calls associated with different data types in the main run logic.

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.

Various pros and cons should be considered when distinguishing if C++ OOD or a C procedural design approach like that used in PPA3 is better for a program. In OOD, class modularization improves large complex codes by allowing functionalities like inheritance to reuse functions/components without excess code. In addition, class use simplifies error checking by minimizing repetition, thus allowing for code expansion with minimal rearrangement of other code segments. C++ has additional safety guards such as encapsulation techniques that prevent coders from mishandling fields and unintentionally altering values. In the project, the adaptability and focus on object template design over sequential code from PPA3 made using C++ OOD the wise approach.

Contrary to C++ OOD programs, the procedural design approach in PPA3 is simpler for beginner coders to understand their code layout. If class organization is poor in C++ OOD, maintaining program structure is incredibly difficult and can result in memory allocation defects. In addition, coders must ensure that they implement the 4 of 6 properly to prevent potential memory leakage. Although procedural design is simpler, when the code becomes more complex, the program can become unmanageable and prone to bugs as readability becomes compromised.

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.

   Throughout all .cpp and .h files, the team sufficiently commented by explaining the purpose of all functions, loops, and variables. In the objPosArrayList.cpp the team does have a section of commented-out code labeled "taking out for now". Although this does not impede the code's functionality, removing this section will prevent reader confusion since it serves no purpose to the project and spans across 45 lines, thus reducing the code's ease or readability. To improve the commenting I would ensure over-commenting and redundancy do not occur. For example, in the Player.cpp file, the team adds the comment "//wrap around" four distinct times with only five lines of code separating each comment. Saying one statement at the top of the code segment will limit the redundancy.

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.

   Throughout all .cpp and .h files, the team followed good indentation, sensible white spaces, and deployed newline formatting for better readability. The newline and white space controls provide sufficient space to distinguish between differing code segments. To improve the readability, initializing all declared variables at the top of the main file will follow proper coding convention. For example, "int check" and "objPos thisSeg" in Project.cpp could have been initialized at the top of the DrawScreen function instead of in the middle of a code segment.

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)

No buggy behaviour was observed as the code executed correctly according to the program instructions. The randomized foodBucket bonus feature was not attempted, however, the necessary single randomized food item was properly implemented and ensured no overlap with current snake segment positions. One implementation to further improve the code efficiency would be to utilize the isPosEqual function of the objPos class, as multiple x and y positional checks occurred throughout the program file and were completely written out every time. The screen included a lot of flickering, however this cannot be controlled. One recommendation for improved user experience is an additional printed statement that informs users how to quit the game. The exit key is ESC, however gameMechs needed to be accessed to determine this.

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 drmemory report indicates 0 unique, 0 total, and 0 bytes of leaks, thus the program executes proper memory allocation and deallocation.

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 using the Pos struct for x and y values in addition to the symbol data member of the objPos class was sensible. The Pos struct increased the readability of the code and increased memory efficiency through the use of a pointer of data type Pos. However, using a Pos struct for higher dimensions would be more beneficial where many variables and vectors may be included, or when there are structs or enumerations within structs. Structs increase code organization and for the purpose of this assignment implementing the Pos struct didn't hinder the functionality of the code. Despite this, the x and y values could have been implemented as simple data members called via the dot operator to avoid over-engineering.

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>

   An alternative objPos class design involves treating the variables x and y as direct integer data members of the class objPos rather than members of the Pos struct. This implementation is counterintuitive as the position variables x and y will no longer be encapsulated by the Pos struct but rather accessed using the object name and dot operator. Although utilizing a struct increases organization and readability, implementing x and y as simple objPos data members will increase simplicity and eliminate the need to allocate and delete allocated memory for members of the Pos struct in the objPos.cpp file. The changes from a struct to objPos data members can be seen in the UML diagram below. Notice the data members are of type integer and are present in the public scope, although x and y may also be implemented in the private scope. The remaining member functions remain the same, although a destructor is no longer necessary since no memory is being allocated on the heap. Copy constructors will directly copy the x, y, and symbol values via the dot operator rather than with the -> arrow symbol for pointers.

objPos
+ x: int + y: int + symbol: char
+ objPos() + objPos(xPos: int, yPos: int, sym: char) + ~objPos() + objPos(a: objPos& const) + operator=(a: objPos& const) : objPos& + setObjPos(o: objPos) : void + setObjPos(xPos: int, yPos: int, sym: char): void + getObjPos() : objPos const + getSymbol(): char const + getSymbolIfPosEqual(refPos: objPos* const) : char const + isPosEqual(refPos: objPos* const) : bool const