COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members	Remy Robinson & Rishi Chakrabarty
Team Members Evaluated	Kartik Seth & Ahnaf Zareef

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 **30 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.

   Upon reviewing the main program loop, we found that the overall object interactions were clear and logically structured. The responsibilities between the major classes (GameMechs, Player, Food) were separated cleanly, making it easy to understand how input detection, movement updates, food generation, and drawing all connect.

   One thing in particular that I didn't think was necessary in the mainprogram loop was the need to print out the scoreboard section in a for loop.

```
129
130         if (i == 0) // at the top print SCOREBOARD
131         {
132             cout << "   SCOREBOARD";
133         }
134         else if (i == 2) // just two lines below print the Score
135         {
136             cout << "   Score: " << gamePtr->getScore();
137         }
138         else if (i == 4) // another two lines below the score print the length
139         {
140             cout << "   Length: " << player1.getPlayerPos()->getSize();
141         }
142
143         cout << endl;
```

   This same implementation can just be done much easier with much less lines [3 lines instead of 14] if you printed it outside of the loop and included a few "\n" (carriage returns). Consider using MacUILib_printf instead of cout for in-game output, as it integrates better with the library's screen handling.

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 OOD:
   - OOD allows you to have modularity within your designs, which inherently makes it easier to develop additional features and makes it easier to debug, as the code is more organized.

   -OOD allows your code blocks to be both reused or extended, through methods such as polymorphism or inheritance.

   Cons of OOD:

   -In OOD, there is often more code split between multiple functions, i.e., each class usually has a .h file and a .cpp file. In smaller-scale projects, this becomes a hassle to work with.

   -With OOD, there is a larger learning curve than with traditional line-by-line programming. You must have a good understanding of references, DMA, and classes.

   Pros of Procedural Design
   -Procedural design is easier to implement with smaller programs. i.e., For PPA3, it was easier to do everything because everything was handled directly with functions and shared state. Doing this procedurally must be faster than making it via an object-oriented design.

   -Often faster to develop because you have to go through fewer files.

   Cons Of Procedural Design
   -Because there are more global variables in procedural design, when you run into errors, it could be a nightmare to debug.

   -Compared to OOD, procedural design has less reusable code; it requires many more lines in your main program to make a duplicate snake, for example, to test it at a specific size. Vs in the final project, it was much easier to test the snake at different sizes, as there was only one loop in main, you had to declare to make the snake a different size.

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.

   When it comes to commenting on their code, we believe that was very strongly done within all files. If we were given the main program file and none of the other files, it would be extremely easy to understand what each method does and how it plays a role within their program.

```
108     // after player moves create a new snake at new pos
109     snakeBody = player1.getPlayerPos();
110     gamePtr->setSnake(*snakeBody);
```

In this section, the comments basically gave me a clear idea of what these two methods are doing and the role they play in the program.

Another example of their commenting can be seen in the ObjPosArrayList section

```
objPos objPosArrayList::getHeadElement() const
{
    if (listSize <= 0)
    {

        return objPos(); // the return obj so we return the empty object const if its invalid
    }

    return aList[0];
}
```

Where they are clearly describing the purpose of how they are implementing the return for error handling.

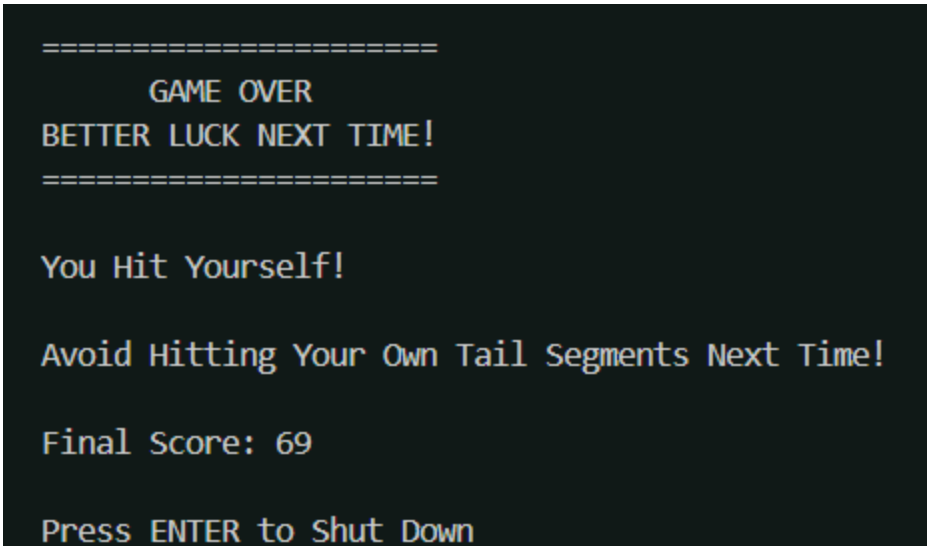
Ultimately, we feel like the commencing made it very easy to follow from a technical perspective; however, we can see that in certain areas, there can easily be some difficulty understanding some code blocks. For example,

```
if (insertedFood.collisionCheck(*snakeBody, snakeBody->getHeadElement().pos->x, snakeBody->getHeadElement().pos->y))
```

Someone from a non-technical perspective would have no understanding of what this call is doing, because it's recursive, but having a simplified explanation of this line would make things a lot easier.

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.

Overall, we feel as if this code has good file composition as indentations are universal throughout all for loops and if statements, there's a valid amount of whitespace between certain functions to make specific features clearer, and they deploy newlines in an adequate amount to make the terminal look clearer. One thing in particular we like is how well spaced out the death screens are in the game, as it makes it much clearer to read.



Overall, we think the file structure in terms of indentation, whitespaces and newlines is very well used throughout the program.

Peer Code Review: Quick Functional Evaluation

1. **[3 marks]** Does the Snake Game offer a 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)

   After trying out the game, we thought that it offered a very smooth playing experience with minimal errors. The game didn't have any noticeable bugs, and I liked how they explained that certain fruit objects were special and showed the symbols of those special fruits. It felt very immersive and engaging. We did not notice any noticeable buggy features while playing the game, but there is one issue that could be fixed in the program. Specifically, in the copy assignment operator, the program does not delete the original pos object before assigning new values. This could lead to incorrect memory handling and may become problematic if the object design changes or if the assignment operator is used more extensively.

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.

This snake game does not cause a memory leakage. This matches our expected outcome, as every heap object has a destructor implemented, which makes it so that there is no leakage.

```
==595== LEAK SUMMARY:
==595==    definitely lost: 0 bytes in 0 blocks
==595==    indirectly lost: 0 bytes in 0 blocks
==595==      possibly lost: 201 bytes in 3 blocks
==595==    still reachable: 83,831 bytes in 156 blocks
==595==         suppressed: 0 bytes in 0 blocks
==595== Reachable blocks (those to which a pointer was found) are not shown.
==595== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==595==
==595== For lists of detected and suppressed errors, rerun with: -s
==595== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

The possibly lost memory leakage occurs in the MACUILIB. The library uses its own memory inside the program, and since you cannot control or clean it up yourself, any leftover memory at the end of the program comes from the library, not from the Snake Game code.

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to yours, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

I believe the compound object design used in the objPos class, where an objPos contains a pointer to a separate Pos struct, is not sensible for this project. In the context of a Snake-style game, each object position only stores three small pieces of data: x, y, and a symbol. Introducing a dynamically allocated Pos struct adds unnecessary heap allocations, extra indirection, and forces the programmer to implement the full rule of four or six. This significantly increases complexity without providing meaningful design advantages.

The purpose of composition is usually to reuse large components, encapsulate behaviour, or separate logical responsibilities. However, the Pos struct in this design contains no behaviour and only two primitive values. Embedding x and y directly into objPos would be simpler, more readable, and more efficient. Because there is no scenario where multiple objects share or transfer ownership of the same Pos, the pointer-based design complicates memory management for no real architectural gain.

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.</u>

Since the compound pointer-based design is not ideal, I would improve the class by removing dynamic memory entirely and storing the coordinates directly as value members. This eliminates deep-copy requirements, removes manual memory management, and naturally supports safe copying through compiler-generated constructors and assignment.

The compound object design used in the original objPos class is not helpful in this project because the additional Pos struct and dynamic memory add unnecessary complexity. A more sensible and maintainable design would store the positional data directly as value members within the objPos class. The improved UML model demonstrates a cleaner and more intuitive structure that aligns with typical C++ object-oriented practices.