COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members	ghosha20@mcmaster.ca	manojv3@mcmaster.ca
Team Members Evaluated	maklaa1@mcmaster.ca	jensenjj@mcmaster.ca

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 logic within the main program loop clearly demonstrates the interaction between different objects, such as **Player**, **objPosArrayList**, and **GameMechs**. The code closely follows object-orientiated design principles as every object interacts with one another to correctly handle various aspects of the program relating to the game mechanics, collision detection, and player position. This makes the code easy to follow and very straightforward. The game also provides a unique message based whether you quit early or die by hitting yourself.

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:

- Classes can be used to encapsulate different aspects of the game logic which helps to improves modularity and clarity
- Helps to distinguish the responsibilities and roles of each section of the program, helpful for debugging
- Allows the code to be reused for future changes such as to develop new features like more complex game mechanics

Cons of C++ OOD:

- Prone to memory leaks if not addressed properly
- Debugging object interactions can be challenging
- Depending on the scope of the project it may add extra complexity to a code, making the development process more difficult than normal procedural design

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 includes a lot of comments that clearly explain the logic and functionality behind key functions and its relation to the rest of the game. It's clear that the group made a strong effort to document their thoughts process

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 practice, an appropriate amount of whitespace and follows newline formatting. There are only a few small areas with some inconsistent spacing, particularly within the nested loops. However, for the most part, the code is very well organized and readable. By just adding a few more spaces and indenting, the code would be fully consistent.

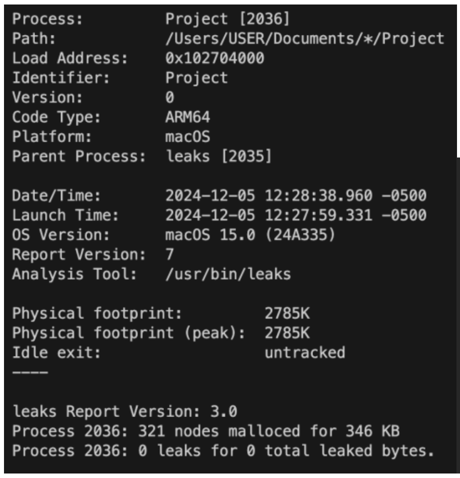
Peer Code Review: Quick Functional Evaluation

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 game runs smoothly and fully demonstrates all the gameplay mechanics for a snake game. The interactions between the snake, the food, and the board are well-executed. There were no bugs or buggy features during our testing of the code. It passed our testing criteria and ran and ended perfectly every time.

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.

No, according to both XCode and Dr memory there were no memory leaks at all.

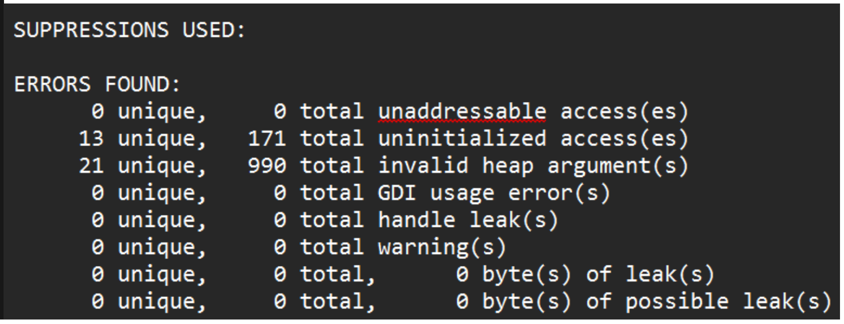


```
Process:         Project [2036]
Path:            /Users/USER/Documents/*/Project
Load Address:    0x102704000
Identifier:      Project
Version:         0
Code Type:       ARM64
Platform:        macOS
Parent Process:  leaks [2035]

Date/Time:       2024-12-05 12:28:38.960 -0500
Launch Time:     2024-12-05 12:27:59.331 -0500
OS Version:      macOS 15.0 (24A335)
Report Version:  7
Analysis Tool:   /usr/bin/leaks

Physical footprint:         2785K
Physical footprint (peak):  2785K
Idle exit:                  untracked
----

leaks Report Version: 3.0
Process 2036: 321 nodes malloced for 346 KB
Process 2036: 0 leaks for 0 total leaked bytes.
```



```
SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
     13 unique,   171 total uninitialized access(es)
     21 unique,   990 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      0 unique,     0 total,      0 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
```

<u>Project Reflection</u>

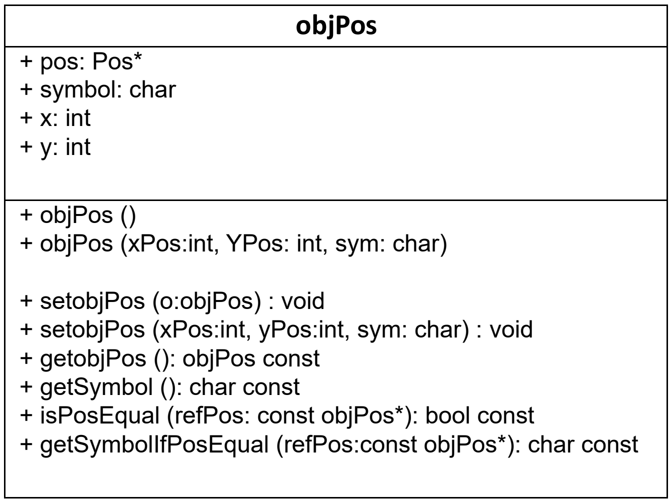
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

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

After reflecting on this project, we believe that the compound object design of the **objPos** class is not sensible for this project. It does some benefits as it encapsulates important information such as position data as well as other information like the player symbol. This structure also makes the code a little bit more straightforward, however, this would cause problems when trying to access the values x and y as it would require memory to be allocated and require the use of pointers to access crucial information about the snake body. This increased the chances of possible errors in the code as well as potential memory leaks.

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 would be to remove the Pos struct and store the values of x and y as integers within members of **objPos**. Now that x and y are within the class, x and y can be accessed using various getter and setter methods. This also removes the need to allocate memory as it does not need a pointer to access the values.



```
objPos
-------------------------------------------------------------
+ pos: Pos*
+ symbol: char
+ x: int
+ y: int
-------------------------------------------------------------
+ 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
```

As per the UML diagram, the values would be within the class **objPos** which would allow them to be accessed more freely and easily.