COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members **karrayb** **rakinm**

Team Members Evaluated **(team a)** **ademb** **zhouz272**

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 names of the functions and methods they use make things more clear. For example, when drawing the snake, they assign a variable referred to as 'segment,' followed by methods named 'getX()' and 'getY(),' which makes it clear what they are doing. The logic is solid and very intuitive. No complaints.

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:

- Code is so much more organized
  - Minimizes confusion in the main file, allowing for easy code following
- Code can easily be reused without repetition.
- Structure is relevant to the programming structure used in the industry, allowing students to gain experience.
- Very good approach for more experienced programmers who are tackling large projects.

Cons:

- Requires a lot more setup before anything can get done (such as the Rule of 6 Minimum 4).
- Memory allocation can get confusing (harder to keep track of what needs to be freed and where.
- Makes debugging a lot more difficult to do.
- Overall more difficult for beginners to grasp

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 is well commented in the main logic. The comments in the draw function make it easy to tell what loops conditions are drawing which parts of the board. However, when it comes to the classes, the comments are sometimes lacking in detail, leading to some confusion about how the logic that they mentioned is actually being implemented. An argument could be made about whether some of their comments are even necessary, since some of the method purposes they try to comment on can just be determined from the actual names of the methods.

the comments aren't necessary, since the method's purpose they're trying to comment can almost always be ascertained just from its name.

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 team does a very good job at indenting and adding sensible white spaces and are overall consistent with how they implement the curly brackets of methods and conditions. One small piece of advice would be to also change the main program function bracket positions as well as the method prototypes to match how theirs are positioned, as those have the opening bracket positioned on a new line after the definition, while theirs place the brackets on the same line as the method/condition definition. This is something very small, but would be something worth considering for the future, when they get involved in large projects in the future, whether they're personal or related to their work.

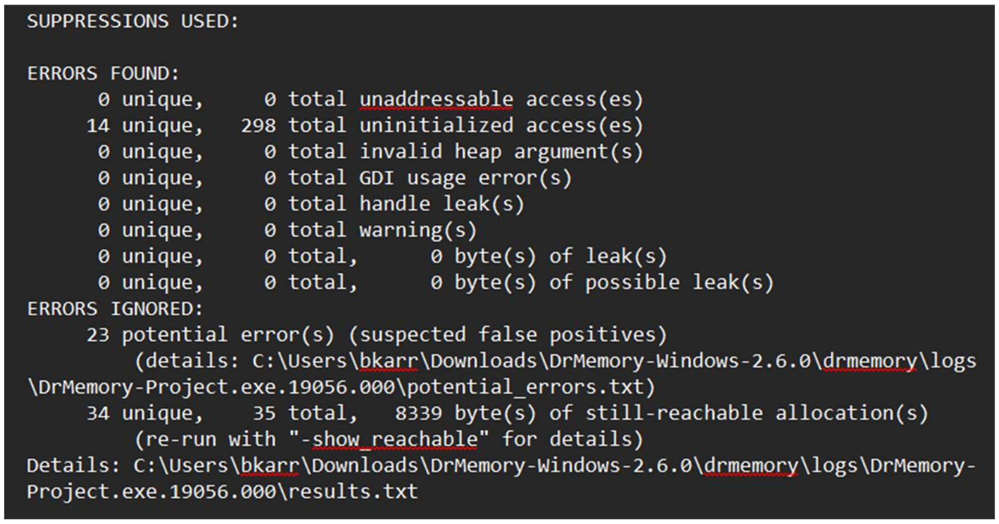
<u>Peer Code Review: Quick Functional Evaluation</u>

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 code appears to have a decently sized bug, which is that the snake can travel through the borders of the game board. This was discovered through our testing of the executable, where a perfectly timed input change seemed to cause the snake to disappear, only to reappear once a new input was processed. By printing the current position of the snake after each loop, it was identified that it was travelling in the borders. The root cause of this is due to incorrect handling of the wraparound. The team can debug this by doing what we did to actually identify the issue, which is just tracking the current position of the head of the snake. Besides that, and the jittering effect observed in almost every executable of the completed project - regardless of the author - the code appears to function as expected.

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.

After performing a Dr. Memory test, it was observed that there was no memory leakage. Here is a snippet of the report.



```
SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
     14 unique,   298 total uninitialized access(es)
      0 unique,     0 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)
ERRORS IGNORED:
     23 potential error(s) (suspected false positives)
         (details: C:\Users\bkarr\Downloads\DrMemory-Windows-2.6.0\drmemory\logs
\DrMemory-Project.exe.19056.000\potential_errors.txt)
     34 unique,    35 total,   8339 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
Details: C:\Users\bkarr\Downloads\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.19056.000\results.txt
```

After further investigation, those uninitialized accesses were from the MacUILib, and therefore the project members are not at fault.

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:

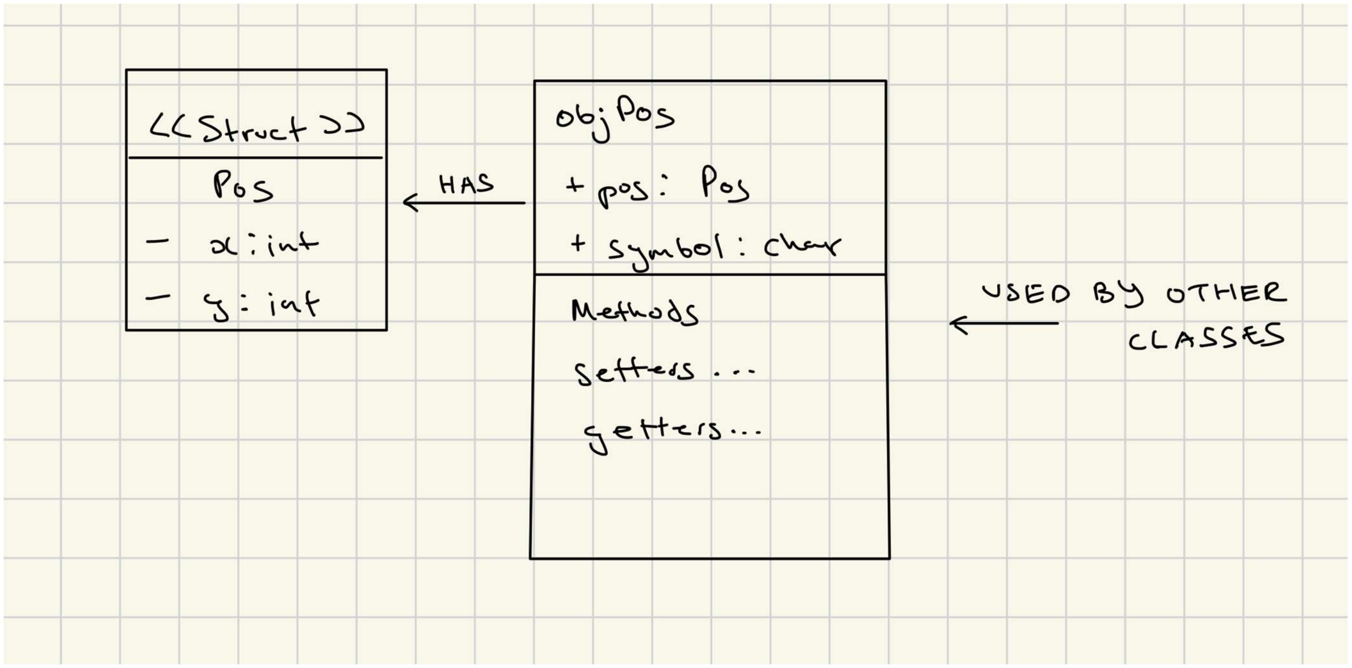
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

   While having a dedicated class for each snake body is very organized, the implementation of objPos is very near perfect with the fatal flaw of having to dedicate heap memory for the "pos" variable. We are allocating memory on the heap for a struct element, this is much less efficient than just letting "pos" be equal to a struct of typedef Pos. We run into many more overcomplications

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>

   I would improve the object design by defining "pos" to be a typedef Pos struct instead of a pointer initialized in the heap. This would use the same amount of memory as defining "pos" on the heap but instead it will do so on the stack and will automatically destroy the memory without the need of the destructor.

The objPos can then be reconstructed as the following (including the getters and setters):



Simplifying the “pos” variable to a struct would allow accessing the struct elements much easier, this could in return simplify the code by eliminating the need for setters and getters. In addition, there would be no need for the destruction, as mentioned above, and no potential of memory allocation problems.