

COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members bojcevg and dcostd1

Team Members Evaluated micham19 and lebely1

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 main function is very well organized, and it is easy to interpret how each object is interacting with the others. The comments used by this group also help to explain certain sections of code. Each line is descriptive enough to explain what it is doing in relation to the other objects, and nothing is redundant which makes the code simpler as well. One thing that could be improved is adding a few more comments for the longer lines of code but other than this, the code is written very well.

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 readability
- Makes future expansion of the project easier
- Simpler implementation of the main function/file
- Makes code safer when all of the members are private
- Easier debugging, bugs are connected to features which are isolated to classes

Cons:

- Might make memory manage more difficult if multiple classes own an object
- Sometimes tracing the code might be more difficult since features and their logic are across multiple files

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.

Within each class, every function has a comment associated with it, and if needed, there are additional comments within the functions to explain specific lines of code. One thing I would suggest is adding additional comments on the longer lines of code as they can be hard to follow at times.

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 indentations and breaks between lines help to organize the code and make it very easily readable. However, certain lines of code are a bit lengthy which makes the readability slightly worse. This could be fixed by splitting up these lines by using if statements instead of returning one very long line.

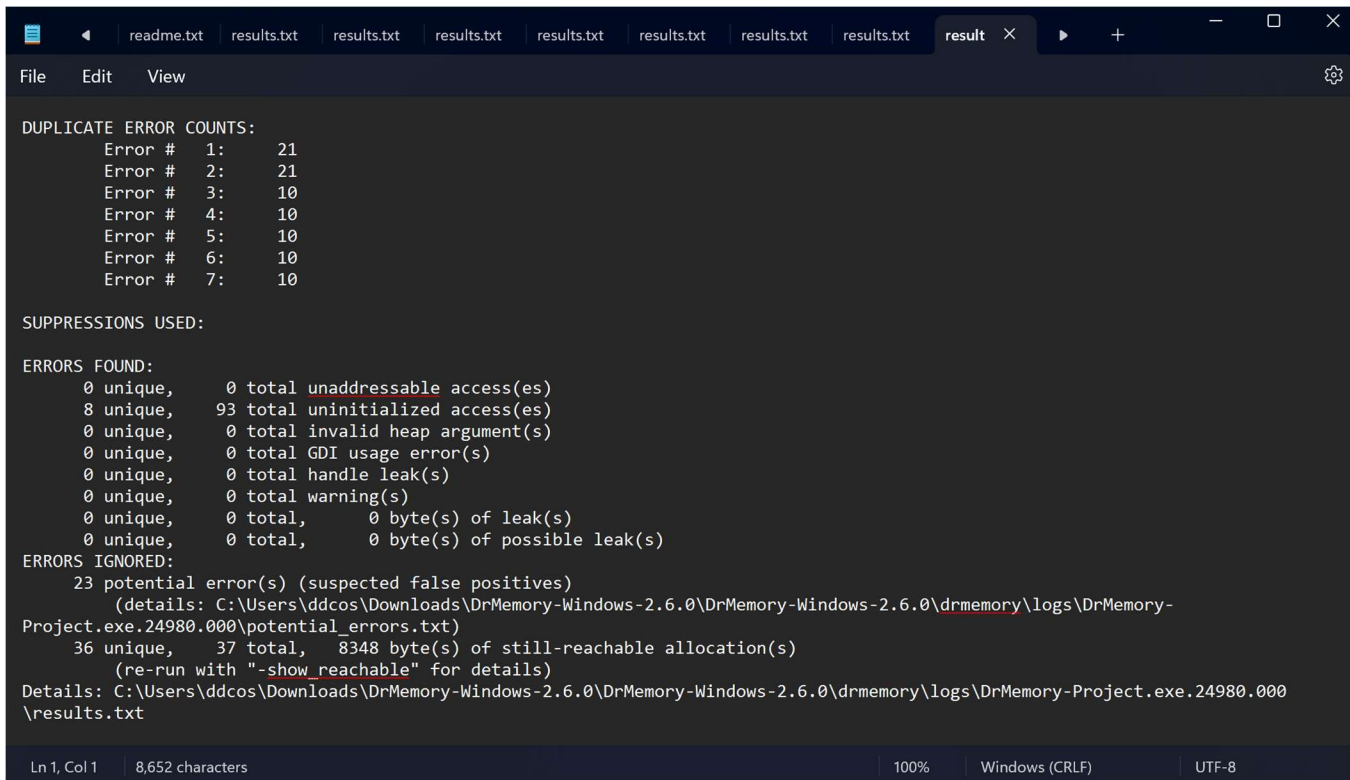
Peer Code Review: Quick Functional Evaluation

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 offers a smooth and bug free experience playing experience. There are no buggy features while playing the game and everything works as intended. One thing I would recommend however, is a slightly upgraded UI which provides game instructions for the user (inputs and food position).

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 running the game, there is no memory leakages.

A screenshot of a code editor window with a dark theme. The window has multiple tabs at the top, with the active tab labeled 'result'. The editor displays the output of a DrMemory analysis. It starts with 'DUPLICATE ERROR COUNTS:' followed by a list of error numbers and their counts. Then it shows 'SUPPRESSIONS USED:' which is empty. Next is 'ERRORS FOUND:' with a list of error types and their counts. Finally, it shows 'ERRORS IGNORED:' with details about potential errors and unreachable allocations. The status bar at the bottom indicates 'Ln 1, Col 1', '8,652 characters', '100%', 'Windows (CRLF)', and 'UTF-8'.

```
DUPLICATE ERROR COUNTS:
Error # 1: 21
Error # 2: 21
Error # 3: 10
Error # 4: 10
Error # 5: 10
Error # 6: 10
Error # 7: 10

SUPPRESSIONS USED:

ERRORS FOUND:
0 unique, 0 total unaddressable access(es)
8 unique, 93 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\ddcos\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.24980.000\potential_errors.txt)
36 unique, 37 total, 8348 byte(s) of still-reachable allocation(s)
(re-run with "-show_reachable" for details)
Details: C:\Users\ddcos\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.24980.000
\results.txt

Ln 1, Col 1 | 8,652 characters | 100% | Windows (CRLF) | UTF-8
```

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?
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).

I think that the compound object design of the objPos with the additional Pos struct isn't the best way to create the class. It makes accessing the x and y positions of the object much more complicated syntactically, and since the Pos object is dynamically allocated, it might lead to more difficult memory management since objPos is an object that other classes have control of so if you're not careful with handling the destruction of the allocated objects leaks are almost inevitable.

To improve the design, I would just simply have the x and y positions be private int members of the objPos class and then add a getter function for each member. This reduces the syntax greatly for accessing them and makes memory management easier.

objPos

- symbol: char
- x: int
- y: int

+ objPos()
+ objPos(xPos:**int**, yPos:**int**, sym:**char**) : void

+ 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**

+ X(): **int const**
+ Y(): **int const**