COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Naram Hirmiz and Mohamed Sabri

Team Members Evaluated: Carson Henderson and Noah Iwasaki

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

Part I: OOD Quality

1. **[6 marks]** OOD is about sensible code modularization. Looking at the header files of each object, can you easily interpret the possible behaviours of the objects involved in the program, and how they would interact with each other in the program? Comment on what you have observed, both positive and negative features.

Yes, the header files are well written and are quite easy to interpret their use and objective. It is evident and clear their use, behaviour and interactability with each other. For instance, the commands snake.movement(), snake.checkSuicide() and apples.checkFoodCollision(snake) make sense and follow a logical sequence. They utilised the matrix efficiently to call on them rather than use pointers which rendered the code much more legible and easier to follow. Additionally, it avoids confusing the compiler too if the pointers have the same name (an issue we faced and had to fix).

2. **[6 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 team members evaluated kept their code legible and simple to follow. Each function called on is adequately named to aid the reader, for instance "snake.checkSuicide()". The logic behind the main program loop is quite simple and clear, while the exitFlag has not changed values the code checks the input, updates the player's position, checks if it has eaten itself, if not it checks if it has eaten an 'apple'. Of course, each function calls on other functions within to update the score, end the game, etc.

3. **[5 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.

The C++ OOD approach in the project calls on each element as an object. Not only does this render the code much more legible and organized, but it is also easier to follow and simplifies the logic. The OOD approach renders our interactive elements as objects in which each has its own position, coordinates and behaviour/functions within classes. This makes it easy to follow and understand. Our procedural design focuses around our functions meaning our code is built around the functions and their outputs since they are not classified and assigned in specific classes so as to not lose track of where they are. The use of OOD helps recreate real life scenarios, for instance in this project, our food is organized in a class however in a procedural design it would not be classified in a class. This can be seen as a both a pro or con depending on the utilization. Since the procedural design is not object oriented, it is more focused compared to object oriented code since as specified previously it is focused on our functions and their outputs. As a result, it is harder to call on such outputs later in the code compared to an OOD approach.

Part II: Code Quality

1. **[5 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 project.cpp file in the code is extremely easy to follow and does not require any form of comments to understand, however, this team went the extra mile to ensure complete understanding of everything going on in the project.cpp file and commented most of it anyway. It is very easy to follow, as they organized every main function of the program with cases and its separate header file, therefore, when calling on them in the main functions in project.cpp, we know exactly what Is going on thanks to the name given to those specified functions. The only addition I would add is a comment in objPosArrayList to further explain the wipe function they added.

2. **[4 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.

It does in fact follow good indentation, there are white spaces and newline formatting between each function to differentiate them, this accompanied with comments facilitates the readability and understanding of the code. Each function follows proper indentation and are easy to tell apart. The same applies to the classes and their header files, as well as the main code.

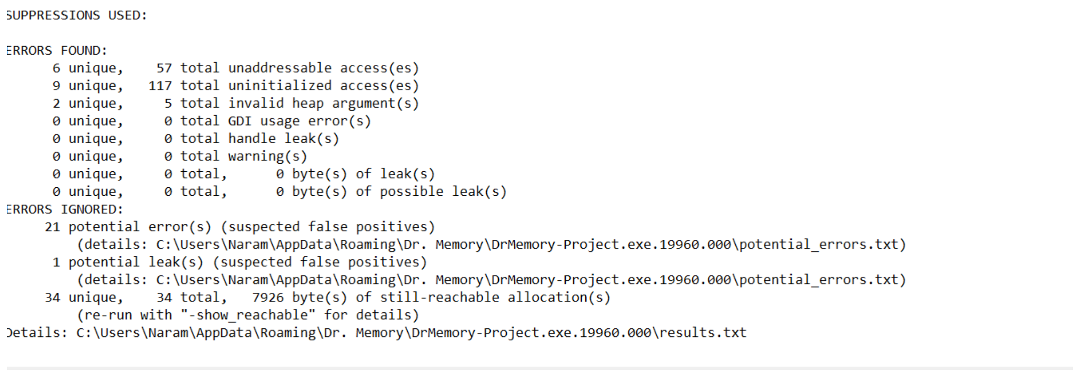
Part III: Quick Functional Evaluation

1. **[8 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 a technical user feedback)

The snake game is smooth, the player's movement is fluid without hiccups. Looking at the code, the only comment or potential risk would be the robustness of the functions in ObjPosArrayList, as they did not include a conditional statement for a case in which a list is 0. However, this case is not common and would not arise since the list is always at a minimum of 1. Therefore, this should not pose an issue unless the game's logic is changed later on. Although this is not a bug, due to the difference between the height and width of the grid and the chosen shape representing the snake, it looks as though a space is created (it skips a column) between each part of the body when moving on the vertical axis. Additionally, due to this, it can be confusing when the player is zigzagging. This is merely a visual discontinuation, this illusion can be fixed by changing the symbol and size of the grid.

2. **[6 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause of the memory leakage.

The Snake Game does not cause any memory leakage according to the drmemory report that was ran. This flawless memory report stems from successfully deleting all arrays and variables that were allocated memory in the heap, in their respective deconstructors and properly calling those deconstructors at the proper time.



```
SUPPRESSIONS USED:

ERRORS FOUND:
      6 unique,    57 total unaddressable access(es)
      9 unique,   117 total uninitialized access(es)
      2 unique,     5 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:
     21 potential error(s) (suspected false positives)
         (details: C:\Users\Naram\AppData\Roaming\Dr. Memory\DrMemory-Project.exe.19960.000\potential_errors.txt)
      1 potential leak(s) (suspected false positives)
         (details: C:\Users\Naram\AppData\Roaming\Dr. Memory\DrMemory-Project.exe.19960.000\potential_errors.txt)
     34 unique,    34 total,   7926 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
Details: C:\Users\Naram\AppData\Roaming\Dr. Memory\DrMemory-Project.exe.19960.000\results.txt
```

Part IV: Your Own Collaboration Experience (Ungraded)

1. Tell us about your experience in your first collaborated software development through this project – what was working and what wasn't. If you are a one-person team, tell us what you think may work better if you had a second collaborator working with you.

   It was much more efficient having another developer working on the program in parallel with you as it allowed us to split the tasks and finish faster. Additionally, it helped with debugging as you could get a second opinion and a different perspective on how to do different actions. However, one of the major cons was with a lack of communication it was very easy to accidentally modify your partner's code, especially when merging two commits. We also had to spend some time synchronizing and matching our names to each other's when splitting the work. There were also some worrying moments as we operated GitHub and accidentally pushed wrong commits and broken code.