

# COMPENG 2SH4 Project – Peer Evaluation

Your Team Members                      \_Serena\_Santos\_    \_Kasra Noyan\_

Team Members Evaluated            \_Bill Zhou\_            \_Yuchen Tao\_

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

## Part I: OOD Quality

1. **[5 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.

The objects in the header files were named appropriately and were easy to understand. The structure was straight forward and it was easy to understand the behaviours based on the naming of the methods. To further clarify their header files, they could have added additional comments. They had their include statements at the top of their header files which clearly indicated which files were interacting with each other. They included “#define, #ifndef, and #endif” statements at the top/bottom of their header files so that the related header files aren’t copied more than once. By using indents and a sensible ordering of their methods, the behaviours within each header file were easy to interpret. Overall, the header files were organized and easy to follow, but could have been further clarified by adding some additional comments.

2. **[5 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.

Generally, the following the logic in the main program loop was quite intuitive. One feature of their code which could have been simplified for easier interpretation would be their run logic method. The code here could have been more concise if the checkFoodConsumption() and checkSelfCollision() logic was instead placed in the player class (lines 71-88 of Project.cpp). Once again, adding additional comments and also spacing would have made the code easier to follow. Furthermore, in the ‘if’ and ‘else-if’ statements in runLogic(), there are equality checks with values returned by the checkFoodConsumption() function and the numbers 1 and 2 (respectively) (lines 71 and 77), but these values are not explained or intuitive. When tracing it back to the player class, these values are still not straightforward (although we can figure out that they two different actions for a regular and special food). This could have been clarified by using comments or variables with variable names to represent the values and their meanings.

3. **[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

- Increased readability due to naming of objects/methods
- Easier for multiple people to collaborate on OOD code by assigning different classes to different people (modularized)
- The fact that the code is modularized allows certain classes to be reused for various use cases
- OOD is more secure because certain data can be hidden

#### Cons

- Tracking the interaction between different classes can become difficult to understand or keep track of, whereas in procedural design this is a nonexistent issue
- C++ OOD is slower and uses more memory because the compilation process is more complicated, whereas C procedural design has a much more straight forward compilation process and faster run time

## **Part II: Code Quality**

1. **[4 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 generally deploys a sufficient self-documenting coding style based on the naming variables, methods and objects throughout the program. As mentioned above, there are certain parts of the code (such as the checkFoodConsumption() equality comparison) that could have used clarification by assigning sensibly named variables to represent the values. As for the comments, there was a lack of commenting throughout the entirety of the program. Providing comments throughout the code to explain the functionality/behaviour of the methods within each file would have allowed for much easier code readability.

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 had appropriate indentation throughout, including indenting within if statements, loops, and the public/private sections within header files. However, some of the spacing and newline formatting throughout the program could have been altered to improve the organization and readability of the code. For example, when initializing variables (or pointers to variables) in Project.cpp, it is good practice to group related/similar variables and space out different ones.

For example, in Initilization() in Project.cpp...

Before:

```
objPos tempPos;
myFood = new Food();
myGM = new GameMechs();
myPlayer = new Player(myGM, myFood);
objPosArrayList* tempPlayer = myPlayer->getPlayerPos();
myFood -> generateFood(tempPlayer);
```

Suggested Changes:

```
objPos tempPos;
myFood = new Food();
myFood -> generateFood(tempPlayer);

myGM = new GameMechs();

myPlayer = new Player(myGM, myFood);
objPosArrayList* tempPlayer = myPlayer->getPlayerPos();
```

## Part III: Quick Functional Evaluation

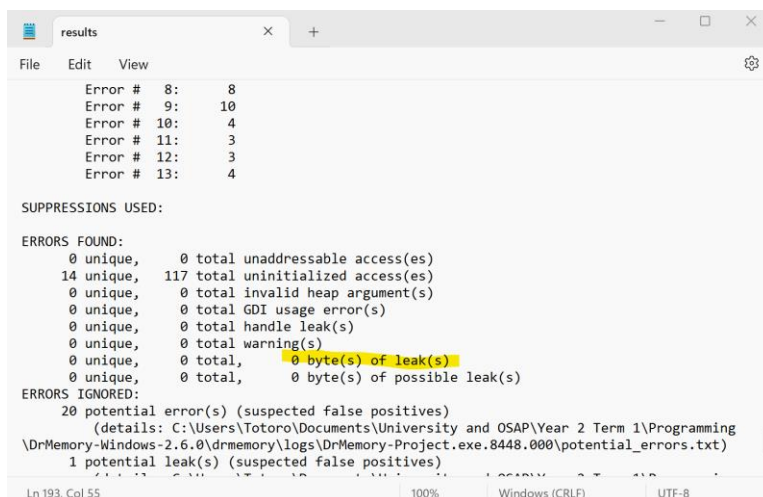
1. **[6 marks]** Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just a technical user feedback)

The games overall functionality in terms of movements, wraparound for the borders as well as the snakes "growth" seems to operate smoothly. In addition, the random generation of food throughout the game board operates correctly each time food is consumed. After self-collision, an end game message appears with the final score and an indication that the game is over. In terms of issues with the code, the first noticeable issue is that the score for the game is initially a large random value. This is likely a garbage value in memory that was caused due to the fact that they did not initialize their variable which represents the score before adding to it. The solution to this would be to initialize their variable "score" in the constructor for game mechanics (GameMechs).

Despite this, the incrementation of the score seems to be operating correctly because the value increases by 1 for a normal food consumed and by 10 for special food. However, the display of the game does not include any instructions for controls or explanation of the symbols within the game. There is also no indication of the exit key. This could be solved by displaying that WASD represents the directional movement, that space bar is the exit key, and by stating that 0 represents normal food (+1 length and score) while 1 represents a special food (+10 score).

2. **[4 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.

After running dr memory for the program, the report showed 0 bytes of memory leaks as shown in the screenshot below:



```
results
File Edit View
Error # 8: 8
Error # 9: 10
Error # 10: 4
Error # 11: 3
Error # 12: 3
Error # 13: 4

SUPPRESSIONS USED:

ERRORS FOUND:
0 unique, 0 total unaddressable access(es)
14 unique, 117 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:
20 potential error(s) (suspected false positives)
(details: C:\Users\Totoro\Documents\University and OSAP\Year 2 Term 1\Programming
\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.8448.000\potential_errors.txt)
1 potential leak(s) (suspected false positives)
```

The reason why there were no leaks was because they had delete() statements in their destructors within their classes which allocated memory on the heap. Specifically, these are the Player.cpp, objPosArrayList, Project.cpp and Food.cpp files.

#### **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.

The collaborated software development experience worked well over the course of the project. Initially, there was a learning curve to figuring out how to use GitHub properly with multiple users within one repository, but after this was clarified it became easy to use. The fact that we were given separate roles and iterations to develop made the design process easier and more efficient than working alone. Overall, we worked better as a team to finish the project at a good pace.