COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Qusay Qadir, Ryan Brubacher

Team Members Evaluated: karthic, mohammed

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.

The header files of each object do accurately reflect how each of the classes function and relate to each other. This makes the code easier to understand, as well as condensing the code because it can be reused in multiple other files. We were able to easily interpret how each class interacted just from viewing the header files. The public member functions as well as the private member variables are all properly named so that they relate to what their function is inside the class. The only issue we could find was that in the *Player.h* file, there are unnecessary member variables called *tempbody* and *playerPos* that do not appear to be referenced anywhere else in the code and may just be left over from earlier iterations. There is also the *moveCount* private variable that does not have a getter function and is not utilized anywhere in player, which may make it unnecessary. These things could make reading the header file more confusing. Other than these, all the member functions and variables, both public and private are properly named and labeled so that it is easy to read and understand the code.

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 main program loop is well organized, and all the objects are created, initialized and destroyed together so it makes it easy to follow. Inside the logic portion of the program, the objects are used in simple ways that make it easier to understand, and all of the created variables are well named to the objects that they relate to. Examples of this are the use of *tempBody* and *tempFoodPos* to store values for the Player and Food objects. I could not find any negatives with this approach as it greatly shortened the code and made it more readable. It ensured that there were less mistakes after making the initial class, as every object made using it follows the same template. Finally, it allowed the reuse of code so that I immediately knew what an object was when it was initialized.

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.

Pros:

- Objects and classes provide modularity that make it easier to manage and organize code throughout the project compared to PPA3.
- Encapsulation allows you to reuse classes in other parts of the code or in future projects. This makes it less likely to encounter errors, as long as the original class is correct.
- The OOD approach provides a clear distinction of entities, increasing code readability and understanding and allows for easier debugging.
- Private and public members encapsulate specific variables and functions, preventing unintended interference and simplifying access.
- An OOD approach makes it easier to build off the classes in future iterations, as well as modify what is already there.
- Creation and destruction of the object are also much easier using an OOD approach, because they have dedicated creator and destructor functions.

Cons:

- OOD concepts had a steeper learning curve compared to the procedural design approach in PPA3. It initially took more time to understand and implement.
- There is also a higher risk of memory leaks, because a lot if the classes use dynamic memory allocation that may be more complicated to delete after use

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.

After looking over the cpp files for objPos, objPosArrayList, there seem to be no comments included around the functions to help illustrate to the user what it does, and how does it relate to the main project file. There are minimal comments in the player files, and they do not help to explain the code very well. There are, however, a good number of comments in the project file that work to explain the code and functions include. They adequately explain the creation and destruction of variable, as well as what their purpose is. There are comments on loops explaining what they do as well.

Overall we think that the shortcoming observed from the code quality can be better improved if the class files that have a large application for the functionality of the game can also be commented such as Player.cpp, objPos.cpp, and objPosArrayLis.cpp. This would help the reader understand how each variable interacts and what their purpose is.

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.

The code provided does provide reasonable and sensible indentation, as well as empty lines for better readability. The space between lines of code does not seem exaggerated and can be seen to be effective in improving the readability of the code. The code can also be observed to have simplified the logic and produce non-redundant lines of code to offer better readability and understanding of the code. All of these factors contributed to making the code easier to understand and read through.

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)

There are a few problems with the snake game. Firstly, the wraparound feature is not implemented correctly, with the snake skipping over spaces at the X borders and printing over the border symbols and the Y borders. After analyzing their *movePlayer* function, they mixed up the X and Y borders when making the snake wrap around. They could work to debug this by ensuring they are referencing the correct border and to extensively test their game after each change. The next bug was that there was no food spawning, both at the beginning of the game and throughout. I would suggest for them to print debugging messages such as the foods position on the board to see if the problem is from the generation or the printing. After quickly looking through the main program, the lines used to print the food are incorrect because of an extra bracket. There is also no code that allows the snake to grow, so even if the food was initialized and printing, then the snake would still never grow. The final bug that I found was that there was no button to exit the game before you lose. I would encourage them to add a button that sets the exit flag so that you can properly exit the game without forcefully terminating the program.

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.

I was not able to run the dr. memory initially because their code does not have a dedicated terminate button. After modifying their code to have an exit key, there does appear to be some memory leakage, specifically from the generate food function in the GameMechs class. After analysing the function and created variables, I found the cause. The leak is because they dynamically allocate memory for a temporary playerPosList to reference while generating the food, but then they never delete the playerPosList after use. This results in a memory leak of 12 direct bytes and 2400 undirect bytes, as stated by the dr. memory report. They should instead not create a temporary variable for the player position and should instead just use the position passed into the function that already references the player position. That would completely avoid having to allocate new memory inside the function and would avoid all leaks. This was the only memory leak found by the report, as the properly deleted the classes after use in the clean_up function of the main file.

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 fun.