PEER CODE REVIEW

Submitted by: Ayomiposi Osho
Student Number: 001427504
Lab Section: L05

Submitted by: Ayaan Khan
Student Number: 400459403
Lab Section: L08

Team Members Evaluated:
Timothy Luo & Ethan Aita

COMPENG 2SH4
Submitted on: December 5th 2023.

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.

Perusing the header files, the object behaviours are easy to guess. This is due to the class data members and functions having descriptive titles, enough to understand what individual constructors or functions do.
With respect to class interaction, each '#include' helped us see the interactions between h files. The pointers in the food.h and player.h were self-explanatory due to their titles; this clarified how information was being shared between files. There was also a comment to explain members created, and the rationale as to why they were made public rather than in the private scope.

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.

DrawScreen function:
Their choice of variable names was self-explanatory and helped in understanding what they are referring to. The comments were a great guide in understanding the flow of their logic, clarifying their step-by-step decisions.

RunLogic function:
This was very brief as it was 3 lines long. The first 2 lines were self-explanatory due to the variable names selected. There were no comments. The 3rd line could've had a comment just explicitly stating that from game mechanics, they used the clear input function. It was still easy to interpret regardless of the lack of comments. The brevity allowed for ease of understanding.

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:

- Procedural programming created code that was lengthy and somewhat disorganized when compared to C++ OOD. In C++ OOD, the classes in their individual h file helped keep things organized and compartmentalized so that you knew where information about each subject is kept and how to access it.
- To change something in the code, you will go to the class and change a few things. Whereas, in procedural programming you will go through the entire code, find errors, and then correct them. Error correction is time wasting in procedural programming.
- C++ OOD requires thoughtfulness prior to implementation. The logic must be thought out first before anything can be done. It prevents crash coding.

CONS:

- A lot more variables and functions to keep track of in their different header files which added complexity to it. The classes have properties compared to procedural programming which involves step by step instructions.

- You can read procedural code from top to bottom and understand it. C++ OOD requires flipping between files to connect the dots, but once connected, can be understood. It is more work to understand the layers in C++ OOD.
- It takes time to create the logic and relationships in C++ OOD with respect to the classes, their members and functions.

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 code is very well commented. You can read the comments and understand the flow, and functionality of the code. There were comments in all the c files (step by step) and main program, explaining each function of loop or if statement. It also wasn't cumbersome.
The comments summarized the purpose of the code.

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.

PROS:
The code is well indented, on all c files, project, gamemechs, food and objpossarraylist. There are intentional spaces between the code that help segregate the code into small steps or function by function. The spaces make the logic easier to understand because what works together is kept together and there are spaces between different functions to make it clear to the reader the individuality of the functions. New line formatting was well used in the code as each new line marked the begging of something e.g. function, while loop, data member etc.

Display Screen: We liked that the positions of each snake segment was evident whilst playing the game.

CONS:
Player.cpp was quite lengthy and indentation was off at line 91 and 206 by 1 space character. It might have been better to avoid the new line spaces just for ease with respect to readability e.g. line 39 – 48 could have used 2 less lines just to bring things together and not make the page look as lengthy. The white space was not necessary in some lines, e.g. line 71.

Display Screen: The BoardSize, I would have placed above the game board. The Score should be placed on its own line just for ease of readability for players.

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 technical user feedback)

We did not notice any buggy features. Playing the game was a smooth experience.
For user feedback, it would be easier for the special food to have a symbol that's not as similar to the normal food because it made it harder to discern which was which.

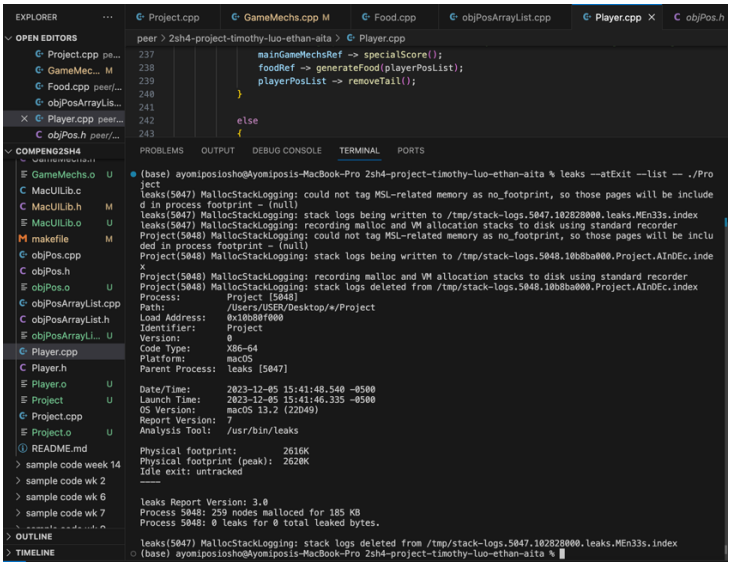
The game was played at a slower pace (i.e., 0.2s delay) to observe each character of the game and no issue was seen. For each time the snake bit itself the game loss statement appeared "You lose! Your final score is 'x' ". For each time the special food was consumed, the score was incremented, and the snake size did not increase. For each 'esc' pressed, its respective statement was printed "Game Ended". Everything worked as it ought to. There were 2 ways to exit the game, by the 'escape key' or a loss; both methods had different print statements when they occurred. Implementing an escape sequence was nice because you are not forced to play the game until a loss occurs. The code was simply written and straight forward. It is clear that they chose the straightest path with respect to logic implementation.

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(s) of the memory leakage.

The Snake Game did not cause any memory leaks. This team made sure to delete all items placed on the heap, i.e.,

i. foodBucket was deleted with the Food destructor created.
ii. aList was deleted with the objPosArrayList destructor.
iii. playerPosList was deleted with the Player destructor.
iv. myGM, myFood and myPlayer were deleted in the cleanup function.

Here is a screenshot of the memory report proving that the team had no memory leaks.



Your Own Collaboration Experience (Ungraded)

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.

Posi:
I think it was a great learning experience working as a team. It forced me to understand the actual purpose of github with respect to creating code in a team. We implemented git pull/clone more times than I had in this entire semester.
In my opinion, we tried to balance the workload as much as we could, recognizing each other's strengths and weaknesses. We both respected each other deadlines, understanding that there was other coursework we each had to also spend time on.
We always ran stuff by each other before making final decisions. We compromised when required.
It was also a plus that we had the same operating system. Otherwise, it would have been a mess commenting and uncommenting certain things every time the code was pulled or cloned between partners. I think Operating systems should be a huge factor considered when pairing teams up for the project.

Ayaan:
I thought this project was a good accumulation of all the PPA work we had done throughout the course, it felt like a good conclusion where I could actually involve the work that I had done previously. In terms of a collaborated software development, this project worked very well, my teammate and myself could both work parallel to each other on milestones and it still working effectively, a product of OOD. I did have issues learning how to use some GitHub commands but that can be summed up to inexperience. The milestone iterations were well designed in my opinion, compared to the PPAs where I really never looked at the recommended workflow, this project really required the use of milestones when working with a partner.