COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Timothy Luo, Ethan Aita

Team Members Evaluated Ayaan, Ayomiposi

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.

Most of the header files, classes, and methods are properly named and easily describe what their functions are, allowing us to easily interpret their intentions. However, there are some cases where this is not true. For example, in Player.h, the method increasePlayerLength takes a parameter int x. However, this parameter is not named appropriately because from the header file, it is not obvious as to what this parameter represents unless you look in the source file. There is one potential minor critique in the GameMechs.h file. They have many flags, including a winFlag. However, we were under the assumption that there is no winning condition in the snake game, which would make this flaf redundant with loseFlag. (In their main program file, they have a win condition set to a score of 2000, however, they also comment that it is a purposely unreachable score, which would in fact make it redundant)

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.

For positives, their idea of printing the gameboard using a 2D matrix was a clever and concise alternative to the standard method of using many nested for loops with flag logic. This change made it easy for them to configure the gameboard and removed the need for us to try and interpret the logic of many nested for loops.

For negatives, there was some improper OOD practice in their program. Their RunLogic function was very crowded, and incorporated features which should have been in DrawScreen and movePlayer. For example, they are increasing their player length and checking for food consumption inside of RunLogic. This crowding of the RunLogic function is bad OOD practice and breaks down their code into something more procedural than it should be. In our opinion, the main program code in the Project.cpp is not organized well (even disregarding some of our OOD critiques). For example, the generateFood function is first called in RunLogic instead of being part of the Initialize routine. Things like this decrease code readability and make it difficult for us to interpret their features.

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

- The code is more readable and better organized.
- Modularization/parameterization makes it easier to correct code.
- Composition allows for them to use methods from other classes.
- The code is more reusable since methods are separated and each their own specific uses.

Cons

- Must have multiple files open to look through.
- In the moment, it can be less intuitive for developers than procedural programming.
- Can lead to overcomplexity/confusion such as in their runLogic function

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 code is mostly commented sufficiently and does a good job explaining what each part of the code is meant to do. Thanks to this, it is easy to understand their intentions and thought process while they were coding. However, we had a minor issue with the commenting in the GameMechs.h file. They commented "Custom" by their custom methods instead of commenting what those methods were used for. This means that if we wanted to understand the logic behind adding a "winFlag" when there already is a loseFlag, we have to go and look around in the source file instead of immediately getting an explanation. To improve this, they could just add a comment in the h file explaining why they needed to add a winFlag.

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 follows good indentations and has sensible white spaces. Each section of the code is distinctly separated from other different sections so that it is easy to tell which chunks of code are meant to do what. They use new lines well so that the gameboard instructions are all very readable and they came up with a clever idea to print the win/lose statements inside the gameboard, which bypasses the need for extra new lines.

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 is one bug in the code. When the game starts, the score that is displayed is a garbage number. Although the score increments properly, they forgot to initialize score to 0 in the constructor, which leads to this garbage number. To debug this issue, they could put breakpoints in the GameMechs.cpp file to observe when the score variable is given a garbage value. Other than this one issue, we could not find any other issues within the code, leaving us with a smooth playing experience.

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.

```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       1 unique,     1 total unaddressable access(es)
~~Dr.M~~       9 unique,    33 total uninitialized access(es)
~~Dr.M~~       1 unique,    57 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total GDI usage error(s)
~~Dr.M~~       0 unique,     0 total handle leak(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
```

There is no memory leak in the Snake Game (see above). They properly allocated and deallocate everything on the heap in their code and implement destructors when necessary. When game mechanics, player data, and food data were used on the heap in Project.cpp, the proper amount of memory was allocated in the initialize routine and deallocated in the clean-up routine. Dynamic memory allocation was also correctly performed in each of the class definitions inside of constructor and destructor methods within the player, object position array list, and food classes.

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.

Overall, we had a good experience collaborating with each other in this project. Initially, we had some issues with updating our repository consistently so we could both work on the project simultaneously. However, it was easy for us to communicate with each other, and we resolved these initial concerns. Although we mostly separated our work and each completed different sections individually, we made sure to explain and communicate our changes to each other, through comments and by working together in person. Our main concern with our collaborative experience was that we felt there was not enough work to do in this project for us to work on it separately. A lot of the framework for the project was given to us and we felt that although we split up the remaining work evenly between us, too much time was spent trying to figure out project basics like git commands and dr.memory compared to when we were actually applying our in class learning.