COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Justin Shin and Eric Augustinepillai

Team Members Evaluated Amhar and Andrej

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.

   Looking at the header files of each object, we observed that the code was well formatted, easy to follow, and as a result we were able to interpret the possible behaviours of the objects and their interactions within the program without any issues. Objects of different types (i.e., constructors, setters, getters) were separated by a line, so we were able to clearly distinguish their functions. Additionally, thanks to the comments, the interpretation of the behaviors and interactions of the objects came with no difficulties.

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.

Examining the main logic in the Project.cpp file, we were able to interpret the object interactions in the program logic without any issues.

Codes with similar functions, for example, heap initializations, player movement, and getter methods were grouped together and separated from the others by a line. Above most of these “blocks” was a line of comment that briefly described the functionality, which allowed us to locate ourselves in the logical process. There were also descriptive comments throughout the program that certainly helped the interpretation and following of the logic.

The names of the variables and functions were kept simple and descriptive, which allowed for an easy interpretation of the object interactions and behavior in the code.

Overall, the code is well-organized, written concisely, and well-commented, allowing for an easy interpretation.

In the Player.cpp file however, we noticed in their updatePlayerDir() function, that myDir == STOP is unnecessary in the switch cases.

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:

1) Break down the program into smaller objects. Makes the program more organized and easier to understand, implement, and modify.
2) The objects, once defined, can be reused in the program with a simple call, which saves time over having to rewrite the same code repeatedly.
3) Protection of internal implementations and variables. In procedural programming, everything was left out in the open, which was susceptible to typos and different errors. With C++, object definition and implementation is encapsulated within the separate object files, which protects it from unintended changes.
4) If you are making a project that keeps secure information, OOD is much better as it keeps information private and out of reach from the main stream of code.

Cons:

1) Increased file count.
2) It can get much more complicated very fast.
3) Similar variables may need to be repeated across multiple files. (Ex: needing to make a gamemechs pointer in both player class and main project file)

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 comments are very helpful in understanding the code, and they are worded quite well in allowing the most amount of understanding in as little words necessary. Variable names are pretty good and self explanatory (tempfood, not_overlap). You could use more variables to represent things like the board size and lengths. (int the rand sections, you use 28 and 13 directly for the board size width and height -2) You could have done the same thing as the player class and use a pointer to gamemechs to make the food class more integrated.

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.

All functional statements use proper indexing. The food generation has pretty good newline use for how many statements there are there.

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)

   Overall, the game runs very well and the screen is very smooth in its updates. All the movements work properly, and they register very well. There is only a little roughness when trying to switch to the opposite direction quickly. Sometimes, it may not record the two quick inputs exactly. This is in my opinion a really hard thing to debug as there are 100s of different places where that slowness can erupt from (mostly due to the roughness not being consistent in when it happens). It could be the delay function, the way the has char function works, and where you call and the order in which both the get char and has char function is in. It could also be the computer being slow, but then the whole screen would appear buggy if that were the case. In my opinion, using gdb debugger will take a century to debug this, so I recommend putting print statements in the player class to verify which direction the player is going, and another print statement in the get input function to see if the input is getting put properly. If one of those fails, then you know where the issue is. If they don't fail, then it is probably the order in which you call the functions, clear input function included.
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.

   It does not appear to cause any memory leakages. Although drmemory does not work very well on windows, it says there are no leakages and when looking at the code, it appears that all allocated arrays/pointers are properly deleted so there should not be memory loss anyways.

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.

   Working on different classes worked well. It only got a little harder when you had to integrate classes and functions from both members, and you sometimes may not fully understand how the other one works.
   We adjusted by communicating during each stage of development, always making sure that we were on the same page. This ensured an efficient collaboration dynamic within the team.