COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Matthew Jarzynowski & Shanka Weeratunga

Team Members Evaluated: Sania and Uthra

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

Repo Link: *https://github.com/COMPENG-2SH4-2023/2sh4-project-sania-and-uthra*

Part I: OOD Quality

1. **Shanka [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.

   Each class appears to encapsulate specific functionalities, which is clear based off the names 'Food', 'Player', etc. This helps to understand what each object and class is doing and the relationship between each, especially when they rely on another, like how 'GameMechs' includes 'objPos' and 'objPosArrayList'. The only thing that might help is the addition of more comments in certain classes. For example, the 'Food' and 'objPosArrayList' classes have a good amount of comments, but 'Player' could have used more explanation due to its complexity. Lines like this "*Player(GameMechs* thisGMRef, Food* thisFood);*" or "*objPosArrayList *playerPosList;*". Overall, interpreting the behaviour of the code is straightforward.

2. **Matthew [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 logic of the program seems to follow a similar loop to that of our program. The program starts with initializing the game board through the "Game Mechanics" class, creates a new "Player", and a new "Food" class. It also creates a new "objPosArray" list to store player-based coordinates and generates the first set of food. This shows OOP in action as the main loop is directly calling other "objects" in the program, in which this could be called a positive of the program. Furthermore, in the "DrawScreen" function, the program references, or calls the "Game Mechanics" object in terms of getting the board size, the snake size, and the score. Also, it references the "Player" class where retrieving the players position relative to the board, as well as referencing the 'Food" class to obtain food-based coordinates. These are all positive features, as these are individualized classes, and thus follow the basic-principles, and/or logic of object-oriented programming.

3. **Matthew [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

- Allow for a more “modular”, more of a “block” based approach in terms of program design.
- Allow for the clear definition of classes and their related member functions, in terms of modularization.
- Consider program elements as their own separate entities, and bind them together using references, creating an overall easier to understand program.
- Compared to PPA-3, someone without any relevant background knowledge behind this program could better understand the underlying logic due to the separation of program elements.

Cons

- For newer programmers, considered less “intuitive” due to the referencing and class nature of OOP.
- Runtime may be longer, and thus less efficient due to GNU g++ compiler compiling multiple files rather than a single C file and its relevant libraries.
- Inconsistent function prototypes can create an environment that is difficult to debug or otherwise fix semantic errors.
- Compared to PPA-3, someone with say just a couple of weeks would not be able to understand such an OOP approach without a deep understanding of pointers and/or referencing.

Part II: Code Quality

1. **Shanka [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.

   In the grand scheme of things, the commenting was well done. The comments provide a high-level look at how the game works and provide a good general idea. However, when looking deeper at specific lines of code, there are some complex lines or groups of lines whose interpretation would benefit with some comments. For example

   - **In Player.cpp** tempPos.setObjPos(mainGameMechsRef->getBoardSizeX()/2, mainGameMechsRef->getBoardSizeY()/2,'o');
   - objPosArrayList* playerBody = myPlayer->getPlayerPos();

   Although there isn't much to these lines of code, adding a simple comment can make a complicated line appear much simple with less need for interpretation. Importantly, the 'Food' files are very well commented which is especially beneficial since customized and "non-standard" files are the ones that are typically hardest to understand.

2. **Matthew [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.

   Yes, the code does follow the practice of "good indentation" and overall has a sensible and clear structure. Many of the loops within the program employ the technique of "newline formatting", in that there is an extra newline before the previous code statements, thus creating a sense of improved readability. The only shortcoming that I could identity would be in lines 12 – 14 of "Player.cpp", in that the definition of "tempPos" is not standard which someone who has lacks experience in C/C++ may consider a syntax error, which it otherwise isn't. The same can be said in lines 32 – 34 of "Food.cpp", which has the same "non-standard syntactic implementation".

Part III: Quick Functional Evaluation

1. **Matthew [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)

   Running the program on 2 computers, a laptop and desktop, there was no visible bugs and/or logical anomalies. Board generation, food generation, snake movement, and food collection had no noticeable bugs, and scoring would update correctly based on the food type eaten. Wrap-around conditions for the snake worked correctly, even with "high-polling user inputs". Furthermore, both exit conditions worked correctly, but only the exit condition relating to "snake suicide" produced a console message. The exit condition related to hitting the exit key "esc" did not produce an output. This is to be expected, as in the main program loop in "Project.cpp" there is no mention of such an exit conditional message, only related to suicide. To implement this using our "COMPENG 2SH4 programming knowledge" you can create another "exitFlag" that checks if the exit key was pressed, say within the "runLogic" function in "Project.cpp", and use a comparison operator in "CleanUp", also in "Project.cpp" to check these conditions, either "true true" or "true false". Also, for the "exit key" consider just using "esc" rather than its ASCII value, 27, since without the above comment, someone would need to a lookup table to know which key to press.

2. **Shanka [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.

   There were no memory leaks shown in the drmemory memory profiling report. All the elements instantiated onto the heap are appropriately destructed, a testament to the thoroughness of this groups project.

Together 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.

   This project allowed us both to experience first-hand a true "industry-standard" example of software development. It was interesting to see how one of us could code part of the project without knowing what the other was doing on a different section of the program. However, we found that this method of splitting the code made it harder to get a deep understanding of all aspects of the code, since we are only experts in what we implemented respectively. We only have a very general idea of what the other person did without looking under the hood further. For example, debugging a partner's code would be a difficult task if it ever came to it. To add to the collaboration aspect, once we understood how GitHub, and "pulling" works, everything after that was relatively simple. There were some cases where we would receive a console message related to overwriting each others' files, which led us to send them via Discord and let the person who didn't have the message commit the new changes. This error was mainly due to us working on the files at the same time, and not pulling once someone made a change, thus causing an "out-of-sync" repo and producing the console error. We found that using a GUI-based tool such as GitHub desktop, created by GitHub and having the same functionality as GitHub CLI, was better as it allowed us to see exactly which lines were changed and by who. Furthermore, VS Code's built-in Git tool was buggy, which we replaced using the "GitKraken" extension. Overall, this project was useful in introducing us to "collaborative software development in the 21st century" as well as "standardized industry practices", in relation to software developmental practices of interoperable code-based repositories.