COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Vaibhav Gopalakrishnan & Omar Seif

Team Members Evaluated Thanush Jayanthan & Asim Nisar

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 provide a well-structured OOD for the Snake game, allowing for modularization and clear separation of concerns. The Player object class contains all player-related functionalities, utilizing references to the GameMechs class for the game environment and an enum for easy direction representation. The GameMechs class is responsible for global core game mechanics such as the border, score system and inputs. Though additional clarity for the purpose of each member function and variable through comments would be beneficial. Lastly, the food class encapsulates food generation and management logic, featuring a reference with "GameMechs" for the game environment and clear functionality.

One concern about the OOD approach to their project is that the relationship between the Food object and Player object is not clear. The food object does not hold a reference to the player object nor does the player object hold a reference to the food object. This makes it confusing to understand how they implemented the core feature of detecting whether or not they collided with food (eating it).

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 in the Project.cpp file showcases the relationship between the objects in the Snake game. The positive features include a modular design that captures the distinct responsibilities within the Player, Food, and GameMechs objects within the Snake Game. Additionally, the input handling and movement is very clear and easy to understand as it is just method calls to their respective classes. The use of object-oriented principles enhances the readability and maintainability of the code, allowing for clear separation of concerns.

However, there is room for improvement, particularly in the use of global variables like "border" and "printPos," which could be encapsulated within their respective classes to enhance data encapsulation and reduce potential conflicts. Additionally, the “border” 2-D array is completely unnecessary, as in the “DrawScreen” function the position is calculated for each game object, but before it is printed, the symbol is written into the border array then printed from the border array. This is very redundant as they could just skip the middle step and print it directly.

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.

The procedural design approach in PPA3 lead to a single file which was easy to navigate, however the file itself was bloated and each of the game's features were hidden beneath variable declarations, nested if statements, and for loops. Getting the overall the understanding of the function of the game and the process was very hard, unless you worked on the project yourself. In the C++ OOD approach, each of the games main features are encapsulated within its own object, this allows easier understanding and a separation of concerns. Therefore, when we need to change a specific feature of the game such as the player or food objects, we just must go into the respective class declaration and implementations rather than navigate through a single file filled with every other object implementation to search for code.

Additionally having an OOD approach lets us implement features which are built on top of those objects such as "superfoods" much more quickly than the procedural approach in PPA3 since we can rely on composition or inheritance. However, there are some negatives, as mentioned before, navigating through all the files in a OOD approach can be a little tedious however since each feature / object is encapsulated within its class declaration / implementation it's in fact easier to find what you are looking for through file names. One another negative would be the time required to implement features in an OOD approach, as that intrinsically requires you to update the class declaration, then program the implementation, as opposed to just declaring the set of variables for the feature and hacking away at it in a procedural design approach.

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 does offer sufficient comments and implements a self-documenting coding style, however in some parts this is not true. While function names like Initialize, GetInput, RunLogic, DrawScreen, LoopDelay, and CleanUp are self-explanatory, the logic within these functions could use some extra comments. For example, the purpose of certain sections of code within the DrawScreen function is not immediately clear, such as the handling of the border, food, and player positions. Adding comments that explain the purpose of these sections and the logic behind certain decisions would significantly improve code readability and ease of understanding for anyone reading over the code like us.

In Food.cpp, the code is well-structured, and easy to understand, however (as a small nit-pick) the character 'O' representing the food symbol is hardcoded; it might be better to define it as a constant variable. In Player.cpp, the use of a global variable (food and myGM) through the extern keyword to hold references to the food and gameMechs object is very awkward and could lead to errors. It's generally recommended to avoid global variables and instead pass these references through the constructor arguments and have the references as member pointer variables.

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.

Overall, the code follows good indentation practices, uses sensible white spaces and deploys newline formatting for better readability. Furthermore, it has consistent indentation and the usage of the proper spaces. However, to keep a consistent coding style and improve the readability of the code, it's better to use the same commenting style throughout all the files. In the code, sometimes the comment symbol ‘//’ was followed by a space, other times it was not.

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)

The Snake Game does indeed offer a smooth, bug-free playing experience. However, some areas of the game could be improved, although this is now just preference, I believe it could increase the quality of the game. For instance, the scoring system might benefit from more depth and complexity (the bonus features). Also, the inclusion of the game board size underneath the game is not really necessary. Overall, the Snake Game provides a great user experience, though small changes could enhance the game.

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.

No, the Snake Game does not cause a memory leak. After using the leaks CLI tool on MacOS, there was 0 bytes lost to memory leaks. However, when using Valgrind on Linux to evaluate we saw 201 bytes of possibly lost memory, however this was expected as the MacUILib library causes these leaks when drawing to the screen on Linux and Valgrind detects this operation as leaks. As this is not an error on that the team caused, this still counts as no leaks.

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.

We collaborated pretty well throughout this project, and there were many things that went well, which contributed to the overall positive experience. We had excellent team communication, which made it easier for everyone to understand the project's timeline and what we had to finish. Also, regular team meetings at Thode Library and messages let us make sure we were both productive. This made it easier for us to make sure that each of us had a clear role and responsibility. But, sometimes when we tried to set up a meeting time for our project, we realized pretty quickly that we had our own obligations such as exams and assignments. However, we did push through and decide on meeting times that worked for both of us, and we eventually got the project finished with.