COMPENG 2SH4 Project – Peer Evaluation

Your Team Members

Ryan Phiby Mathew, Ishaan Malhotra

Team Members Evaluated

Elizabeth, Saad

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.

Observing the header files, it is obvious what each file's functionality is and how they tie with each other. The variable names and methods make it clear what their purposes are. Each header file is properly formatted with the variable and function prototypes.

All of the function names are indicative of their purpose. For example, new functions introduced in the GameMechs.h, such as getScore() and printMessage() are very telling in what their purpose is. The use of descriptive variable names in the header file also enhances code readability. For instance, if there's a variable named playerPosition or foodPosition, it conveys its purpose without the need for additional comments.

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 of the Snake Game demonstrates a clear interaction between the three main objects: GameMechs, Player, and Food. The code is structured logically, and the interactions are comprehensible. The Initialize function sets up the initial state of the game, creating instances of the GameMechs, Player, and Food classes. The GetInput, RunLogic, and DrawScreen functions are then repeatedly called in the main loop to handle user input, update game logic, and render the game screen.

Positive features include the use of well-defined functions in the Player class, such as updatePlayerDir and movePlayer, which encapsulate specific functionalities, contributing to a modular and readable design. The DrawScreen function efficiently renders the game elements,

including the player's body and food positions, making it easier to understand the game state visually. Additionally, the code employs meaningful variable and function names, enhancing readability and understanding of the program's functionality.

However, a potential area for improvement is the lack of encapsulation in the Player class. Direct access to the playerPosList from outside the class might lead to unintended modifications. It would be beneficial to provide getter methods or make playerPosList private, allowing controlled access. Overall, the main program loop effectively captures the game's logic and interactions, but minor refinements, such as encapsulation in the Player class, could enhance the code's validity and maintainability.

Another area of improvement would be the collision logic in the DrawScreen function. As the DrawScreen function is used to print the border and player elements, the conditionals used for the collisions with the player element should be in the RunLogic function, to increase code readability and organization.

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 of OOD:

- Modularity and Code Organization
    - Files are specific to their functions
    - The Player.cpp file contains the code related to the player character, such as direction
- Inheritance
- Data Encapsulation
    - Private data in classes
    - Strengthens data security and prevents any accidental errors and corruption
- Updating Code
    - Adding more elements related to the classes already incorporated is easier due to the separation of files and independency of function

Cons of OOD:

- Much more complex than C procedural design
    - Much more code to keep track of
- Verbosity
    - The same effect could be achieved with fewer lines of code and one file in procedural programming
- Harder for maintenance, many more files and lines to go through to find root causes

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 generally includes comments, with almost all files having excellent readability. Positive aspects include the presence of file headers explaining the purpose of each file, as well as comments accompanying function prototypes and class declarations. However, several classes lack comments for their member variables, which could be beneficial for understanding their roles in the code.

The main game loop and input handling functions lack comments, and providing a brief explanation of their purpose would be helpful for a reader unfamiliar with the codebase. Furthermore, error handling comments and messages in critical areas, such as memory allocation, is notable and provides additional context for the code.

Overall, the code does an excellent job at documenting its functions and has high readability, with a low chance of incorrect interpretation.

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 generally follows good indentation practices, uses white spaces in sensible spots, and deploys newline formatting, contributing to readability. All blocks, loops and conditional statements are indented consistently, enhancing the code's visual structure. There is consistent use of spaces around operators, making expressions and statements easy to read.

However, there are a few areas where improvements can be made. In some places, indentation is inconsistent, possibly due to copying or formatting issues. Ensuring uniform indentation throughout the codebase would improve its overall readability. Printed output statements have sufficient whitespaces and newline formatting for easy readability. Overall, the code mostly adheres to good indentation and spacing practices and is very easy to read.

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 mostly offers a smooth experience, with no major flaws in logic. However, there was a border wraparound issue that we uncovered when testing out the game wherein the player icon can be hidden behind the border.

A possible root cause of this problem is in the DrawScreen function in the Project.cpp file, where the border might not be accounted for when drawing the player. Another possible root cause is in the Player.cpp, where the enum cases for directions might not handle the border correctly.

A potential debugging approach would involve reviewing the boundary conditions in the movement logic of the Player class, ensuring that the snake's position is properly constrained within the game border. Additionally, it would be beneficial to validate the collision detection mechanisms between the snake and the game border in DrawScreen.

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.

The Snake Game does not cause any memory leak. When running Drmemory, the report states there is 0 bytes of memory leak, meaning that all the memory on the heap was correctly deallocated. The deallocation that occurs in the CleanUp function is responsible for taking care of any memory leaks by deleting the references to the other classes, myGM, myFood and myPlayer.

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 collaborative software development project proved to be a highly integrative and intuitive endeavor. Throughout the project, we engaged with fundamental programming concepts such as modularization, Dynamic Memory Allocation, and Object-Oriented Design. The project preparation activities not only equipped us for the main project but also guided us through the essential concepts, fostering a deep understanding.

The transition to Object-Oriented Design posed a challenge, particularly when coupled with more intricate concepts such as classes. Nevertheless, we successfully navigated through these complexities and ensured the program's functionality.

One notable challenge arose during the integration of code from our PPA's, transitioning from C to C++. However, with focused concentration and optimization, we swiftly addressed and resolved the issue.

Another hurdle emerged with Dynamic Memory Allocation (DMA) concerning the objArrayList. By closely following lectures, tutorials, and tutorial videos, we promptly identified and addressed the issues, further solidifying our comprehension of the topic.

In conclusion, this project has been a truly rewarding experience and a fitting way to conclude the semester. Witnessing all components of our program seamlessly come together and having the opportunity to play the snake game we created brings genuine joy to our faces.