COMPENG 2SH4 Project – Peer Evaluation

Your Team Members _____Saad_____
_____Shounak_____

Team Members Evaluated _____Kevin_____
_____Hemant_____

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.
    a. The code employs clear and descriptive names for classes and functions, facilitating easy interpretation of the objects' behaviors. Class names, such as GameMechs, objPosArrayList, and Player, convey their respective purposes. Function names like generateFood and updatePlayerDir are indicative of their functionalities. The use of the Dir enum in the Player class adds clarity to direction representation. Overall, the code establishes a solid understanding of object behaviors, and minor refinements can further improve clarity.
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.
    a. The main program loop demonstrates effective coordination between objects, particularly the GameMechs and Player. The code is well-structured and easy to follow User input handling is straightforward, and graphics rendering is well-executed using MacUILib functions. Suggestions for improvement include implementing if the exit character or cheats character is clicked within the gameMechs object.

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.
    a. **Pros**
        i. Encapsulation: OOD facilitates encapsulation, allowing for the bundling of data and methods into cohesive classes, enhancing modularity.
        ii. Abstraction: The use of classes and objects provides a level of abstraction, allowing for a more intuitive representation of entities in the game.
        iii. Clear Object Interaction: Objects interact in a well-defined manner, making the code more readable and helping to separate concerns.
        iv. Member Function Names: the member function names in the classes, such as generateFood and updatePlayerDir, are clear and aligned with their responsibilities

b. **Cons**
    i. Complexity: Object-oriented design can introduce additional complexity, especially in smaller projects where a procedural approach might be more straightforward.

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.
    a. Clarity and Purpose of Comments: The code includes comments, but they vary in usefulness. Some comments explain the purpose of code blocks well, such as in the Food::generateFood method, while others are redundant or uninformative, like //delete heap members.
    b. Use of Self-documenting Code: The variable names and method names are generally descriptive, such as getBoardSizeX and generateFood, which is good practice. However, there are instances, like myDir = STOP;, where the purpose of the variable or its potential values are not immediately clear.
    c. Consistency: There's inconsistency in the approach to commenting. Some methods are well-documented, while others lack comments altogether.
    d. Improvements:
        i. Consistent Detailed Comments: Ensure that every method and complex code block has a comment explaining its purpose and functionality.
        ii. Clarify Variables and Constants: For variables or constants with less obvious purposes (like myDir), add comments explaining their role and possible values.
        iii. Explain Complex Logic: In more complex methods, such as movePlayer, include comments that guide the reader through the logic, especially where multiple conditions and loops are involved.

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.
    a. Indentation: The code generally follows consistent indentation, which aids in readability. However, there are a few places, especially in nested loops or conditionals, where the indentation could be more consistent to improve readability.
    b. Whitespace Usage: The usage of whitespace is mostly appropriate. However, in some lengthy methods, additional line breaks could help separate logical sections of the code.
    c. Improvements:
        i. Strategic Whitespace: Introduce blank lines to separate logical chunks within methods, making it easier to follow the flow of the program.
        ii. Group Related Code: Keep related lines of code closer together, with minimal newlines in between, to visually represent their connection.

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)

   Based on thorough testing and the statement provided, it can be concluded that the Snake Game offers a smooth and bug-free playing experience. All functionalities appear to be working as intended, with no buggy features reported. This indicates that the game has been well-designed and rigorously tested, which is a testament to the robustness of the code and the effectiveness of the testing procedures implemented by the development team. As it stands, the game's performance and lack of bugs reflect positively on the team's programming practices and their ability to create a reliable and enjoyable user 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.

   Upon running Dr. Memory on the executable file of the snake game, no memory leaks were detected. This appears to be due to the meticulous implementation of memory deallocation within each class and the main file, where the delete operations are executed as required. Additionally, the use of destructors, contributes significantly to the program's ability to operate free of any memory 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.

Shounak: My first project collaborating on code with a classmate using GitHub was great. GitHub's tools made it easy to communicate and review code. We divided tasks efficiently and simultaneously working on different aspects of the project went smoothly.

Saad: My initial experience collaborating with a classmate on a coding project via GitHub was excellent. The platform's features facilitated seamless communication and efficient code review. We allocated tasks effectively, and the process of concurrently working on diverse components of the project proceeded without any hitches.