COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Emilie Stal and Mustafa Shahid

Team Members Evaluated David and Nina

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.**

By looking at the header files, it's apparent that the classes are structured quite well and follow OOD principles effectively. Each class, like 'Food', 'Player', and 'GameMechs' encapsulates its functionalities neatly. The use of clear and concise function names makes the code easy to understand and follow. There are no circular dependencies as all the classes build off objPos, objPosArrayList, and one another in a structured hierarchy. Additionally, the modularity is evident as each class is responsible for a specific aspect of the game, like food generation, game mechanics, and player interactions. No negatives were observed.

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.**

After examining the main program loop, the main project file facilitates a clear interpretation of how the 'Player', 'Food', and 'GameMechs' interact with each other by creating references to each class via global pointers. There is a clear division of responsibilities as each pointer (myPlayer, myFood, myGM) is responsible for a class and utilizing its functions. Additionally, the main program logic is very concise and relies on only necessary function calls to other classes which improves encapsulation and use of OOD. One negative is that there are a few 'magic numbers' used in the Initialize() function for initializing the GameMechs object, myGM, board sizes '30' and '15'. Replacing these with defined constants or variables would improve code readability and maintenance for future uses.

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:

- Modular design
    - Clear separation of responsibilities between classes (‘Player’ for player functions, ‘Food’ for food functions, ‘GameMechs’ for game mechanics functions, etc)
- Object encapsulation
    - Each class encapsulates its functionalities
    - Code is more organized and reusable
- Improved object interaction
    - Objects interact with each other through well-defined mediums
    - Clear understanding of responsibilities and dependencies
- Reusability
    - OOD design supports code reuse
    - Classes can be extended and modified to be used in other parts of the program

Cons:

- Added complexity
    - The need for class hierarchies and encapsulation introduces complexity
    - More organization is needed to keep track of which classes interact with each other, which can become very tricky in large programs
    - Debugging for the project was more difficult as multiple files and classes contributed to one operation
- Learning curve for OOD concepts
    - As OOD is a more complicated approach than procedural design, it was a significant learning curve to apply OOD concepts for this project.

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.**

Answer-1)

After reviewing the team's code, I believe that the self-documentation in the code is clear, as they used naming conventions which are consistent throughout the program which not only improves readability for the user but also allows users to understand the purpose of the code much better.

However, I believe that there is still room for improvement in a few areas. First, in terms of comments they should provide descriptive comments, especially for the function which are a bit confusing/logical, as comments help the user to understand the code step-by-step. Secondly, as they used ASCII numbers for their game such as 64 = '@', they should provide a comment on this as a user won't know that 64 value represents @ in ASCII values. Furthermore, error handling should be taken into account such as in the Food Class file generateFood function they should have provided comments as this function may lead to infinite loops.

As providing in-depth comments explains to the user the purpose of the code as there are few functions which are a bit logical and hard to self-understand, using consistent naming conventions and comments for member variables makes the code manageable and understandable. Finally, they used srand(time(NULL)) to generate random food in food.cpp which is fine but moving it to the main file would make the code neater.

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.**

After looking at and reviewing the team's code, it is generally readable as they used indentation and white spaces which makes the functions and code blocks visual representation better which improves the readability of the code. Moreover, they used a new line “\n” mostly in their MacUILib printf statements which separate specific function calls.

However, there are a few improvements which can still be done to improve the readability. Firstly they could’ve separated code blocks using blank lines such as in the DrawScreen function in the project.cpp file. Secondly, they could’ve limited the scope of the variables by placing the variable declaration objPos tempBody; inside the for-loop of the DrawScreen function which would improve the distinction between the declaration of variables and the body of the loop.

To summarise, the code has appropriate white spaces and indentation (new line), the improvements suggested above would make the code look neater and improve the code readability for a user who doesn’t have background knowledge of the code/project.

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 technical user feedback)**

The snake game is well-developed, but there are a few potential issues which impact the game. Firstly, in the game's border checking which is the wrap-around function when the snake's head reaches the border, there conditional statements should include all possible situations otherwise it may lead to inaccurate positions. To fix this I would use a debugger or printf statements to execute and identify any errors in the location.

Furthermore, the generateFood function in the Food class involves random number generation and checking it against the snake position; if the function fails to find a position, it may lead to an infinite loop. Again we can use printf statements to identify if there's a situation where it would go to an infinite loop. Moreover, when running the snake game, the game should quit if the snake collides with itself, however, that's not happening because the collision detection logic in the movePlayer function checks intersections made within the body, but not all collisions are being detected. To prevent the head from being compared to itself, the for loop should begin at i = 1. To prevent comparing the head with the head, i - 1 should be used instead of i when getting the body element using getElement. To identify this error I would use printf statements or run a debugger and look for the output of each iteration and compare them to my requirement to ensure that the game quits as soon as any element of the snake collides with itself.

To summarize, making the above-mentioned changes would improve the game quality and ensure a smooth experience. The point to be noted is that the game does not quit when the snake collides with itself due to the condition of the for loop; otherwise, the game runs smoothly with no major bugs that would impact 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(s) of the memory leakage.**

Answer-2)

The Dr. Memory report indicates that there are no memory leaks in the game, which means the code effectively allocates and deallocates memory during its execution. The team allocated memory on Heap and then deallocated the memory properly using delete after observing their code in the project. cpp file, they deallocated memory for myGm, myPlayer, and myFood using delete in the Clean Up function. Moreover, in the player. cpp file, they deallocated playerPosList using delete playerPosList. In objArrayList.cpp, they deallocated memory for the list using delete [] aList. As they did free, deallocate, or delete memory on Heap, it caused no memory leakage, which supports the Dr. Memory Report.