# COMPENG 2SH4 Project – Peer Evaluation

| | | |
|---|---|---|
| Your Team Members | Maliha Dar | Mariam Mohamed |
| Team Members Evaluated | Abbiraa Murugathasan | Sanna Ghai |

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

## Part I: OOD Quality

**1. [5 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.**

In reviewing the header files of each object in the program, I notice various aspects of Object-Oriented Design (OOD) that contribute to or detract from sensible code modularization. The positive features include a well-organized modular structure with distinct header files for each class (GameMechs, objPos, Player, and objPosArrayList), promoting code modularity. The encapsulation of private and public members within each class adheres to OOD principles. Constructor and destructor implementations ensure proper object initialization and cleanup.

The header files provide a blueprint for the classes' possible behaviors and interactions. The declarations help convey the responsibilities of each class and how they may collaborate to implement the game's functionality. However, the details of the actual implementations in the corresponding .cpp files would further define the behaviors and interactions in the program.

**2. [5 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.**

In the main program loop, the code organizes the game's functionality by coordinating interactions between objects such as `GameMechs`, `Player`, and various `objPos` elements representing different functions for the game. The loop operates by executing a sequence of functions: initialization, get input, game logic execution, screen rendering, and memory cleanup.

`GameMechs` manages the game board and food generation, while `Player` handles player movement, direction, as well as collision detection. The code incorporates conditional statements to ensure valid boundary checks and accurate rendering of game elements within the board's boundaries.

One notable area for improvement is allowing for the game to continue when the snake catches/ eats it’s first food. Unfortunately, the game ends as soon as collision occurs, and this could be as a result of a problem in the run logic within the codebase. Additionally, there are instances of potential code

duplication, especially in the `DrawScreen` function, suggesting an opportunity for refactoring to enhance both efficiency and readability. Strengthening error handling mechanisms would add robustness to the code, allowing it to handle unforeseen scenarios more effectively.

Overall, the code effectively manages game mechanics and object interactions within the main loop. However, enhancing commenting, optimizing for reduced redundancy, run logic improvement and improving error handling would significantly improve readability, maintainability, and resilience of the code.

**3. [3 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.**

C++ OOD Approach:

Pros:

- Classes promote modularity by encapsulating data and behavior within well-defined objects, enhancing code organization.
- Data members are encapsulated, restricting access and promoting a more controlled and secure design.
- Classes and objects provide a level of abstraction, allowing for a clearer understanding of the relationships between different components.

Cons:

- Object-oriented design requires thoughtful consideration of class hierarchies, relationships, and encapsulation of state and behavior. This made it a difficult transition from procedural programing

C Procedural Design Approach (PPA3):

Pros:

- In PPA3, implementing the code was simpler and was easier to understand
- In procedural programming, data is often directly accessible, which can be advantageous for certain types of algorithms or data manipulations.

Cons:

- Relies more on global variables and functions, potentially leading to naming conflicts and decreased maintainability.
- Lack of classes and objects may lead to less abstraction, making it harder to represent complex relationships between data and behavior.

## Part II: Code Quality

**1. [4 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 contains basic comments that offer a surface-level understanding of certain functions and sections. However, these comments lack depth, particularly in more intricate or complex parts of the code. To enhance comprehension, introducing more detailed comments at critical points within the code, particularly in areas with complex logic or operations (such as GameMechs.cpp), would be beneficial. These comments should aim to explain not just what the code is doing, but also why certain decisions or methodologies are chosen.

Additionally, establishing a consistent commenting style across the codebase, incorporating header documentation for functions or classes, and adopting a self-documenting coding style with clear and meaningful variable and function names could significantly improve the code's readability and understanding.

An example of improvement in creating/adding comments could be in the GameMechs.cpp. If we could improve the comments, we would add more depth to allow for a better understanding of the code. For example, in this code

**Their code comments:**

```
//Generates a new food position
void GameMechs::generateFood(objPosArrayList *blockOff) {
    srand(time(NULL));

    foodPos.x = (rand() % (boardSizeX-2))+1;
    foodPos.y = (rand() % (boardSizeY-2))+1;

    objPos tempPos;

    //checks for overlap with blockOff positions
    for (int i = 0; i < blockOff->getSize(); i++){
        blockOff ->getElement(tempPos, i);
        while(foodPos.isPosEqual(&tempPos)){
            foodPos.x = (rand() % (boardSizeX-2))+1;
            foodPos.y = (rand() % (boardSizeY-2))+1;
        }
```

**Suggested/ improved comments on their code:**

```
// Function to generate food position on the game board, avoiding occupied spaces
void GameMechs::generateFood(objPosArrayList *blockOff) {
    srand(time(NULL)); // Seed random number generator with current time

    // Generate random positions for the food within the board boundaries, avoiding edges
    foodPos.x = (rand() % (boardSizeX - 2)) + 1; // Avoid left and right edges
    foodPos.y = (rand() % (boardSizeY - 2)) + 1; // Avoid top and bottom edges

    objPos tempPos;

    // Check for overlap with blockOff positions to ensure food doesn't spawn on occupied spaces
    for (int i = 0; i < blockOff->getSize(); i++) {
        blockOff->getElement(tempPos, i);
        while (foodPos.isPosEqual(&tempPos)) {
            // If there's an overlap, regenerate a new random position for the food
            foodPos.x = (rand() % (boardSizeX - 2)) + 1;
            foodPos.y = (rand() % (boardSizeY - 2)) + 1;
        }
    }
```

**2. [3 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, utilizes sensible white spaces, and employs newline formatting for better readability. Each code block is indented consistently, enhancing the overall structure and readability. The use of blank lines between functions and logical sections improves code organization and comprehension. Overall, the code demonstrates good practices.

## Part III: Quick Functional Evaluation

**1. [6 marks] Does the Snake Game offer a 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)**

Bug 1: Snake Going into Borders Issue:

Upon analyzing the code, it appears that there is a bug in the Snake Game where the snake object can move into the borders of the game board. This issue likely stems from inadequate boundary-checking logic in the movePlayer function within the Player class. The code does attempt to handle boundary conditions, but there may be inaccuracies or oversights in the logic. To address this, debugging efforts should focus on reviewing and refining the boundary-checking conditions in the movePlayer function to ensure the snake stays within the valid bounds of the game board.

Bug 2: Game termination after first collision

Another identified bug in the Snake Game is the termination of the game as soon as the snake collides with the first object. This issue indicates a problem with the collision detection logic, particularly in the Collision function within the Player class. The code may incorrectly interpret the initial position of the snake as a collision, triggering an immediate game over. To resolve this issue, the collision detection logic needs careful examination and adjustment to exclude the initial position or handle it appropriately. Debugging tools and print statements can be utilized to trace the execution of the collision detection logic and identify any inaccuracies leading to the game over condition.

**2. [4 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.**

Abbiraa and Sanna's Snake Game does not provide any memory leakage, as seen in the screenshot we have included below (Figure 1), indicating "0 leaks for 0 total leaked bytes". This proves successful memory allocation and deallocation and proper use of "new" and "delete".

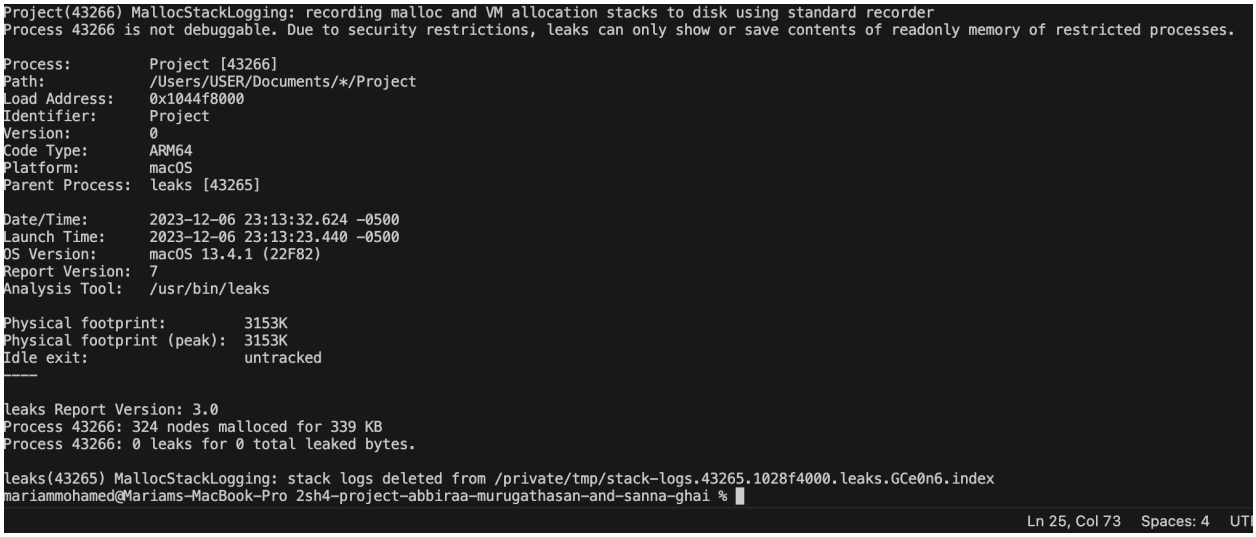


**Figure 1:** Memory Report for Abbiraa and Sanna's Snake Game

## 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.**

In our first collaborative project creating the snake game, there were a few ups and downs, with not knowing where to start, all the way to last minute coding at 2 am. However, although this project was tedious and required a lot of brainpower and sleepless nights, our collaboration and communication skills only got stronger, and we both improved not only as coders, but also as communicators.

Each of us brought different skills, and when we combined them, we came up with great solutions. Our diverse skill sets complemented each other, leading to some interesting solutions that neither of us might have discovered alone. We also spent days in the TA office hours (awesome TA's btw we would not have done it without them lol!), and were able to debug any problems we ran into using trial and error, and every-so-often, using the LLDB debugger. Overall, this was a great learning experience for the both of us, and we are satisfied with the way things went!