COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Raymond Pham	Arabella Paet
Team Members Evaluated	Warun Gumarawell	Yuan Ding

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

Note: The group that we were assigned to did not fully finish the project therefore we decided to evaluate certain parts of their project, and other parts from our own project.

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.

   GameMechs.h includes:
   - objPos.h
   - objPosArrayList.h
     - Based off these inclusions, we can infer that the game mechanics class interacts with some sort of array list consisting of objects on the gameboard.

   Positives:

   - Based on the methods written in the header file, I can understand what the game mechanics class does
     - Generate food for the snake.
     - Capturing input from the user
     - Creating the gameboard
     - Etc.
   - For the generateFood() method, we not understand why they included the objPos and the objPosArrayList classes at the beginning as the method takes in these classes as its parameters.

   Negatives:

   - Difficult to understand how the GameMechs class will retrieve input from the user without the usability from MacUILib.h as it was not included at the beginning.
   - Based off our understanding of the GameMechs class, the class should have a way to control the score of the game, however there is no data member or method that allows the class to do so.

- Even though C++ has the capability of function overloading, it is difficult to understand the need for 2 genereateFood() methods that takes in either a singular object or a list of objects.

<u>Player.h includes</u>:

- GameMechs.h
- objPos.h
- objPosArrayList.h

<u>Positives:</u>

- The method names describe that the class deals with aspects of the Player object:
    - Constructing and destructing a player object
    - Getting player position list
    - Updating player direction
    - Moving the player
- Based on the included objects, I can also infer that the player class builds off-of/ uses methods from game mechanics, objPos, and objPosArrayList.

<u>Negatives:</u>

- The Player class seems to deal with methods concerning the player however the method getScore is not fit to be included in this class if what I assumed for the class is right. Getting the score is more of a game mechanic rather than a player aspect.
- I am unsure on what the getPlayerPosList method is doing with the given parameter. It would be wise to include comments on the functionality of methods or group methods with similar functions together.

<u>objPosArrayList.h includes:</u>

- objPos.h

<u>Positives:</u>

- The structure of the class is very organized with all of the getters, setters, mutators, etc. being grouped together respectively.
- Based on the parameters of some of the methods, we can infer that the class interacts with the objPos class and that it uses that class to create a list of objects from it.

<u>Negatives:</u>

- The data members for the class have an access level of protected which is used for if another class is being inherited from this class. However, it is difficult to understand the need for these data members to be protected since the other classes are not inheriting this class.

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.

   Notes: The main logic of the main program of the group we were assigned to was empty, therefore we decided to evaluate the main logic from our project.

   Our Own Project:
   The RunLogic() in the main program loop is very concise which makes it easy to interpret how the classes and objects are used within the project. Within the main logic, the ptrPlayer object calls onto methods from the Player class: updatePlayerDir() and movePlayer(). One can infer that these methods called by the object controls the player movement based on the users input and on the intuitive names of the methods. At the end of run logic, the ptrGameMechs object calls on the clearInput() method which one can infer, clears the users input from the input stream. Overall, due to the conciseness of the main logic and the names of the methods that the objects are calling, one can interpret the functionality of what is happening within the code block.

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:
   - There are less global variables written in static storage
   - Since there are less global variables, the total memory storage on the heap is maximized
   - less floating variables
   - improved code readability and conciseness (didn't have to keep scrolling up and down a large file to find what a specific variable does)
   - Better user readability (The main program makes intuitive sense when a coder reads it)
   - Very modular since there are different classes that hold specific methods to do certain things instead of a list of function definitions at the end of the program (in PPA3)

   Cons:
   - More files to deal with instead of 1 large file
   - We have to keep track of access level between classes (What is made private/public to outside the class)
   - It was a little more difficult to get a grasp on OOD for C++ than C procedural design in PPA3
   - OOD affects memory performance more than C procedural design as creating objects takes up a lot of memory.

<u>Part II: Code Quality</u>

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.

   <u>Since the other group did not fully complete the project with majority of it being empty, we all decided to evaluate the code quality of our own project too.</u>

   <u>Evaluation of other groups project:</u>

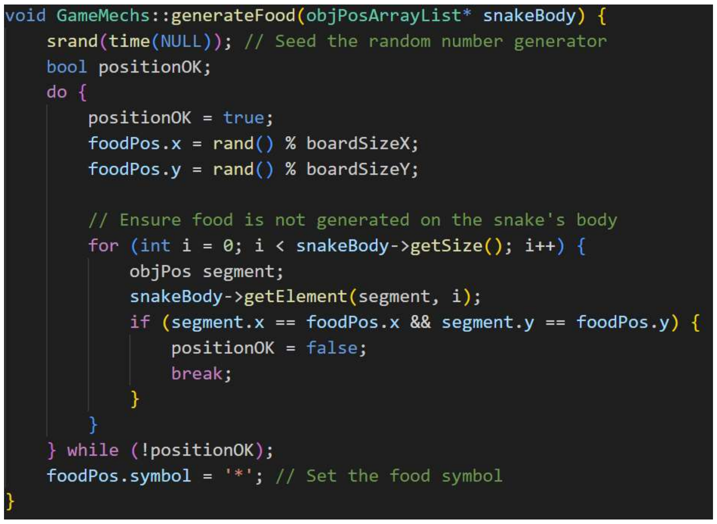
   The code does not contain a lot of comments which significantly made understanding the code more difficult. To improve code readability, more comments should be added, at least to explain non-standard methods.

   <u>Evaluation of our own project:</u>

   The code has a good number of comments that effectively explain portions of the code where it is not immediately obvious what is taking place. The logic for various code blocks was explained in the comments which made it easier to understand how various data was processed. The sufficient commenting also ensured that it is easier for both members of the team to work with the other person's code without having to go through the whole document on each class and method. Something to improve on would be the redundancy of the comments. For example, in the GetInput() function, there is no need to comment what that function does as it is obvious from the function name. Unnecessary comments could clutter the code instead of making it simpler to digest. It is important to keep the comments concise to ensure that it does not negatively affect code readability.

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.

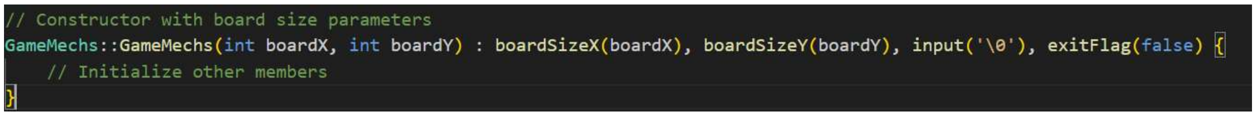
   <u>Other Groups Project:</u>
   From what was observed, the project follows good indention with their code blocks within function definitions as well as for/while loops for better user readability. For instance, the generateFood() method follows proper indentation with loop functions being indented after the conditions as well as the entire function definition being indented to indicate that it all corresponds to the generateFood() method.



```
void GameMechs::generateFood(objPosArrayList* snakeBody) {
    srand(time(NULL)); // Seed the random number generator
    bool positionOK;
    do {
        positionOK = true;
        foodPos.x = rand() % boardSizeX;
        foodPos.y = rand() % boardSizeY;

        // Ensure food is not generated on the snake's body
        for (int i = 0; i < snakeBody->getSize(); i++) {
            objPos segment;
            snakeBody->getElement(segment, i);
            if (segment.x == foodPos.x && segment.y == foodPos.y) {
                positionOK = false;
                break;
            }
        }
    } while (!positionOK);
    foodPos.symbol = '*'; // Set the food symbol
}
```

Newline formatting could have been used for lengthy singular lines of code. For instance, within the GameMechs class, the constructor has multiple parameters which make the code difficult for a user to read without zooming out.



```
// Constructor with board size parameters
GameMechs::GameMechs(int boardX, int boardY) : boardSizeX(boardX), boardSizeY(boardY), input('\0'), exitFlag(false) {
    // Initialize other members
}
```

Overall, for the majority of the program follows proper indentation, sensible white spaces, and formatting for better user readability.

<u>Evaluation of our own project:</u>

- The program follows good indentation with code blocks within function definitions and for/while loops are indented for better user readability. New line characters were also printed after each line within the DrawScreen() function inside main, again for better user readability. Some singular lines of code are quite lengthy as it goes off the screen when a user to trying to read it. For instance, in the DrawScreen() function in main, there is an if statement that checks if the coordinate is on the edge of the board. The if statement has multiple conditions all written onto a singular line. To improve readability, a newline could have been inserted to fit the if statement onto 2 lines instead of 1 continues long line. Appropriate white spaces between characters can be seen within parameters of functions, loops, conditional statements, etc. It helps readability as the code is bunched up together, but instead consists of proper spacing just like a regular sentence should be.

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)

   Note: Even though the other groups project was compliable, we could not test out the snake game. Therefore, we all evaluated the bugginess of our Project too.

   Evaluation of other groups project:

   -doesn't run

   -border just spams, draw function is not implemented correctly

   Upon execution, the program spam prints the border. This is likely due to the draw function not implemented correctly. There is also no way to exit as the program does not take any input. The getInput function does not contain any code as well as the RunLogic function. There is no way to exit the program unless we force exit.

   Evaluation of our project:

   The Snake Game executes and correctly displays the starting screen with the player object in the middle of the screen and the border printed correctly. The food objects are generated at random locations each time one is eaten, and the snake body and player score are incremented based on the food eaten. The snake dies when eating its own body which means the self- collision function is implemented correctly. Proper lose, win, and exit messages are printed depending on the situation. The program flickers quite a bit but this is probably due to the print delay. Some things to improve in terms of user experience is to display some necessary info on the screen. It might not be immediately obvious to the user how to exit the program or what happens when you eat different food objects. To improve this, print statements on important info for the user/player like the exit key, what the superfood does, and rules of 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.

   Note: DrMemory was no able to be done onto the other groups project as there is not manual way of exiting their game. Therefore, we also commented on our own project.

Other Group Project:
In the main program, nothing was deallocated during the CleanUp() routine, which shouldn't be a problem since nothing was created during the Initialize() routine either. However, inside the constructors of the classes such as "Player" and "objPosArrayList", things were created onto the heap. These classes also had destructors that would have deallocated anything the constructors would have created onto the heap. Therefore, if DrMemory were able to run on their program, no memory leak should be seen since they had proper deallocation within the destructors of the classes.

Our Project:
When running DrMemory on the snake game, no leaks/possible leaks were recorded. Proper deallocation was done within the CleanUp() routine in the main function for anything that was created onto the heap during the Initialize() routine. Destructors for classes were also properly implement to delete anything on the heap that the constructor or methods from the class may have created onto the heap.

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.

    - It was difficult at first to understand how to collaboratively work together on a single project. However, with the use of OOD we were able to work on specific portions of the project parallelly by creating different classes/libraries and then combining it in the end. We learned that communication is key when collaborating on a program. The use of comments was very helpful as it allowed each other to understand our assigned coding sections without to having to constantly asking each other what a specific piece of code does.