COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Masooma Naqvi and Michael Bradshaw
Team Members Evaluated: Masooma Naqvi and Michael Bradshaw

Important Note: The team we were paired up with has not completed their project. After speaking with Professor Athar and Professor Chen, we have been asked to critique our own code and answer the questions accordingly.

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 behaviors 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.

From looking at the header files of each object, we can clearly understand the possible behaviors the object will conduct and how the object will work with other objects in the program. For example, in the GameMechs class, we can clearly see that it is controlling the main functions of the game such as the user input, score, playing size, and whether the player has lost. Additionally, we see it holding a food class dealing with food position, generation, and location, which relates to the GameMechs class and functionality. The Player header file shows an example of how the objects in the program are interacting with each other through references and object method calls. In all the classes, we see the use of public and private components ensuring that each component of the object is accessed in the correct manner. We could move the Food class to its own header file so that the code could be isolated better, however the class is so small and it relates to GameMechs so it makes sense why it is in the current location.

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.

In the main program loop, we can see the program following a structured matter with functions controlling each stage in the program such as GetInput(), RunLogic(), DrawScreen(), LoopDelay(), and finally CleanUp() at the end of the program. In each of these function stages, they rely on the interactions between different objects. For example in Initialize(), we see how the GameMechs object is interacting with each of the objects in the program by passing it as a reference. We also see that Food is using the Players object as a reference, indicating that Food will use Players object components to allow it to function correctly. We also see the Player object

and Food object interacting with each other to make sure the food does not print or spawn on top of the player. The interaction between each object follows a structured order making the interaction look clean and easily understandable.

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 are that the code is easy to follow and it is easy to understand how each of the components of the project work and function together. The code allows for easier repeating of certain objects such as food and the player; the OOD approach allows us to do snake extension and retraction easier compared to a procedural approach, which would have us needing to recreate the players properties from scratch. Here we can simply copy and reuse them. OOD allows us to structure and store variables easier and allows for function interaction in the object to be seamless and less cluttered compared to procedural in PPA3. The cons of OOD is that the transferring of data between different objects becomes more complex and requires object data variables to be initialized before interaction, additionally referencing is needed compared to straight inaction between variables. Also more consideration of memory allocation and leak is required to ensure a stable and leak proof program.

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 project has many classes and we believe that some classes have a sufficient number of comments while others do not have enough. The objPosArrayList class has many comments that help the reader understand the code. There were comments explaining when new data members were created on the heap and when data members were deleted. These comments made it easy for us to check if measures were taken to reduce memory leakage. Furthermore, every function with error handling had comments explaining how the throw std:: out_of_range feature will be used if the array list is empty or full. In addition, the insertHead, insertTail, removeHead, and removeTail functions all have comments explaining how the array list will shuffle all the elements to the front or end of the list depending on which function is called. The GameMechs class also had comments explaining variables such as boardSizeX and boardSizeY to help the reader understand their usage in the program. However, we believe that it would be beneficial to add comments on top of the getter and setter functions to help the reader understand which functions are returning values and which functions are changing the values. The Project class also has a number of comments explaining where the player position is obtained and when the FSM is updated. However, it would be beneficial to add a few more comments to the

DrawScreen() and CleanUp() functions to explain how the game board is set up and where the score and player instructions will be displayed. We believe that the Player class needs more commenting. There are a few comments in the movePlayer() function that explain how the user can move up, down, right, and left. However, there are no comments for the checkFoodConsumption(), increasePlayerLength(), and checkPlayerOffGrid() functions. Since these functions include a lot of processing in a few lines of code, adding comments would be helpful for readers to understand how each function runs.

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.

Yes, the code follows good indentation and it has a sensible amount of white spaces. The GameMechs class has excellent organization because there is only one line of white space between each of the getter and setter functions. The objPosArrayList class also has a good amount of white space because each small code block is separated from other code blocks. For example, in the insertHead() function, there is a small code block for error checking, a small code block to shuffle the elements, a small code block to insert the new element, and a small code block to increase the size of the list. If all this code was written without any whitespace, it would be difficult for readers to understand what each code block is responsible for executing. Furthermore, each for loop and if statement in this class has an indentation which is crucial for readability. There is also good usage of newline formatting in the DrawScreen() function of the project class. The player score, player position, player instructions, and ending program messages all begin on a new line which makes it easy for the user to interact with the program.

However, there are some ways that this project could have better formatting. In the Project class, there is a function called DrawScreen() which is responsible for displaying the game board, score, and player instructions. There are a lot of for loops and if statements, but there is not enough white space and the curly brackets to end code blocks have no pattern. Sometimes the curly brackets are placed right after the if statements or for loops and other times, the curly brackets begin on the next line. Furthermore, the Player class needs more white space. The checkFoodConsumption(), increasePlayerLength(), and checkPlayerOffGrid() functions have no space between the lines of code which makes it difficult to differentiate between the for loops and the if statements.

Part III: Quick Functional Evaluation

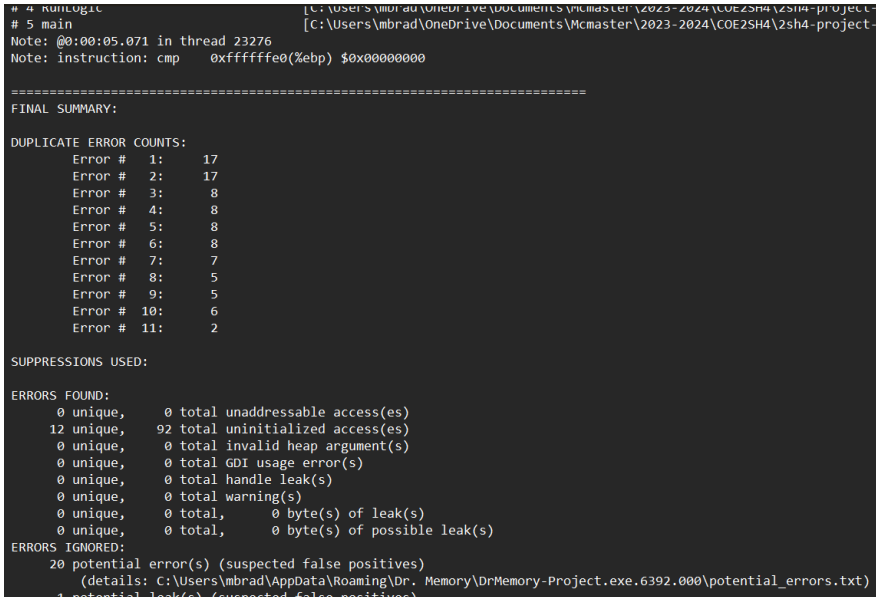
1. [8 marks] Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just a technical user feedback)

The snake game is smooth and provides a bug-free playing experience. However, the snake can be at the bottom of the grid and sometimes it does not show when the left or right button is pressed. This is a bug caused by accidentally placing the player boundary too high. The boundary just needs to decrease by one grid unit to prevent this issue. However, even with this bug, the game still runs without any problems and you can bring the snake up by pressing up or down. The program also displays the location of the player ensuring the player knows where it is. Everything else seems to work fine.

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.

After running Dr. Memory on program, the report shows 0 memory leaks indicating that all heap members and objects are deleted correctly at the end of the program. The classes: Player, Food, and objPosArrayList all feature a class deconstructor where any heap member will be deleted. Player and Food do not feature any delete function calls because they do not create any new heap members. But it doesn't hurt the program to have a deconstructor anyways. The objPosArrayList class constructor contains an instruction deleting the heap objPos array at program shutdown. This prevents memory leaks from happening.

Dr. Memory Report:



```
# 4 RunLogic                          [C:\Users\mbrad\OneDrive\Documents\Mcmaster\2023-2024\COE2SH4\2sh4-project-
# 5 main                              [C:\Users\mbrad\OneDrive\Documents\Mcmaster\2023-2024\COE2SH4\2sh4-project-
Note: @0:00:05.071 in thread 23276
Note: instruction: cmp    0xffffffe0(%ebp) $0x00000000

===========================================================================
FINAL SUMMARY:

DUPLICATE ERROR COUNTS:
        Error #   1:     17
        Error #   2:     17
        Error #   3:      8
        Error #   4:      8
        Error #   5:      8
        Error #   6:      8
        Error #   7:      7
        Error #   8:      5
        Error #   9:      5
        Error #  10:      6
        Error #  11:      2

SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
     12 unique,    92 total uninitialized access(es)
      0 unique,     0 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      0 unique,     0 total,      0 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
ERRORS IGNORED:
     20 potential error(s) (suspected false positives)
        (details: C:\Users\mbrad\AppData\Roaming\Dr. Memory\DrMemory-Project.exe.6392.000\potential_errors.txt)
```

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.

We enjoyed our experience in our first collaborative software development project. This project was an excellent way to combine all the content we learned throughout the semester and apply it to create a popular game that is played among children across the world. Since the project was split up into iterations and each partner was responsible for different parts, it was an excellent way to split up the workload. One struggle we experienced was during iteration 1. In order to complete part A of iteration 1, it was necessary to have iteration B done. However, both of us had different schedules and commitments so it was hard to be on the same page for each iteration. Sometimes we would be behind in the project because we could not move onto iteration 2 or 3 until the previous iteration was complete. To resolve this issue, we made our own deadlines for each iteration and made sure we completed our individual work by these dates. We had excellent communication throughout the project so this was a small obstacle that we overcame. Although it was difficult to apply our knowledge of pointers, and classes while simultaneously learning the content in lectures, we appreciated the help videos and TA sessions because it gave us an idea of how to start each iteration. We enjoyed this project and we look forward to coding more projects like this in the future!