COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Sania Zaman	Uthra Meenakshisundaram
Team Members Evaluated	Jarzynowski	Weeratunga

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.

   Each header file has a meaningful and descriptive name which suggest their respective functionality in the main project. For example, “Player.h” file indicates it likely handles player related operations while the “GameMechs.h” would likely manage game mechanics. The methods within the header files such as “generateFood”, “incrementScore” etc. are provided meaningful and descriptive names as well, providing an idea of the operations that they might perform within their respective class. By grouping the getters and setters for specific members allows for code readability. The methods are also organized in the order in which they are used within the class. For example, in the “Player” class, the “movePlayer” method is mentioned in the end, since it uses other methods within the same class which helps in maintaining code clarity.

   The header files can include more detailed comments for data members, methods, and overall class to explain their functionality and interaction with each other. This helps developers who are not familiar with the code to better understand the class functionalities, data members, and their interactions within the broader program structure. Moreover, it also assists future developers who might work on or maintain the codebase.

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.

   The main program shows the interaction between GameMechs and Player obj by calling the appropriate methods to manage game logic. The meaningful method names allow for code readability and understanding their roles within that specific part. Passing appropriate parameters in the methods shows the communication between different objects and enhances functionality.

The negative feature is the unorganized sequence of the methods in the main loop; DrawScreen is called first, then get input, run logic and than loop delay. This is confusing as Draw screen should be called after getting input and running the logic. Moreover, the get input method uses MacUILib_get char to get the input which should have been done within the GameMechs class getInput method for encapsulation and for a modular design. Additionally, 3 different ways are used to print in drawscreen; MacUILib_printf, cout and printf hinders consistency and code clarity.

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

- Inheriting methods and features from different parent classes allows for code reusability making the overall program more organized, easy to understand and maintain.
- Through polymorphism, the objects can be treated as an instance of their parent class, for example the objPosArrayList is an instance of the objPos and allows to build upon the parent class without changing it.
- Encapsulating the data within the class under private and public prevents the possibility of the data being changed/modified.
- C++ syntax is less complexed than C. For example, c++ uses simple statements for memory allocation. This also allows for easy memory management.
- The code in main program is more organized, readable and easy to maintain and modify.
- Allows for easy debugging as each class can be tested individually as it was done with objPosArrayList class by building testcases.
- Limited code in main program increases program functionality and speed.

CONS

- OOD programing can be complex and very challenging as the coders must know how and when the call the methods from different methods. They also must design the methods within a class based on the methods of different class, deciding on the parameters, return values or the type of the method.
- The amount of code also increases for an easy program due to different classes as they have header and cpp files.

C procedural design approach in PPA3

PROS

- The simplicity of the code without OOD design eases programming and understanding the code

CONS

- Limits programming complex features such as generating special food.
- Leads to unorganized code as more features are implemented within one class.
- Slows down the program leading to glitch in the console.
- Makes it challenging to maintain memory allocation and deallocation.
- Challenging to identify or debug a problem.
- No data encapsulation can lead to maintenance issues.

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.

   Sufficient comments were used throughout to help understand the code. Almost all functions in the cpp and the main file had comments indicating their purpose. Detailed comments were included for the conditional statements, explaining what conditions are being checked. However, the team could have implemented more comments in the header files to briefly explain the members and methods since header files are looked at prior to the cpp files. For examples, explaining that the GameMechs also generates food apart from managing game mechanics. The team included few statements and variables that were never implemented in the overall program. Such as defining the exit flag in the main program but using the exit flag from the GameMechs to run the loop. Moreover, adding a conditional statement to display the quit game method but never implementing a key to quit the game. Overall, the names of variables and methods used were meaningful and the code offers sufficient comments to help understand the code functionality efficiently.

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 follows appropriate indentations, white spaces, and deploys newline formatting for better readability of the code as well as the game board on the console. The comments don't take too much space and most comments are within a line which provides a good balance. We were able to locate a line in the initialize function that could have benefited from adding white spaces after commons that would make code readability easier. An example would be that, in Player.cpp, the conditional statements and the code within is written in one line with enough blank spaces in between that makes the code looks really organized while saving lines of code. The use of newline on the console is very appealing to the user as it introduces the game, score, position, and their names in different lines. The newline formatting between methods further enhances code readability. Overall, there is a sensible and consistent use of white spaces, and the code deploys newline formatting for better readability.

Part III: Quick Functional Evaluation

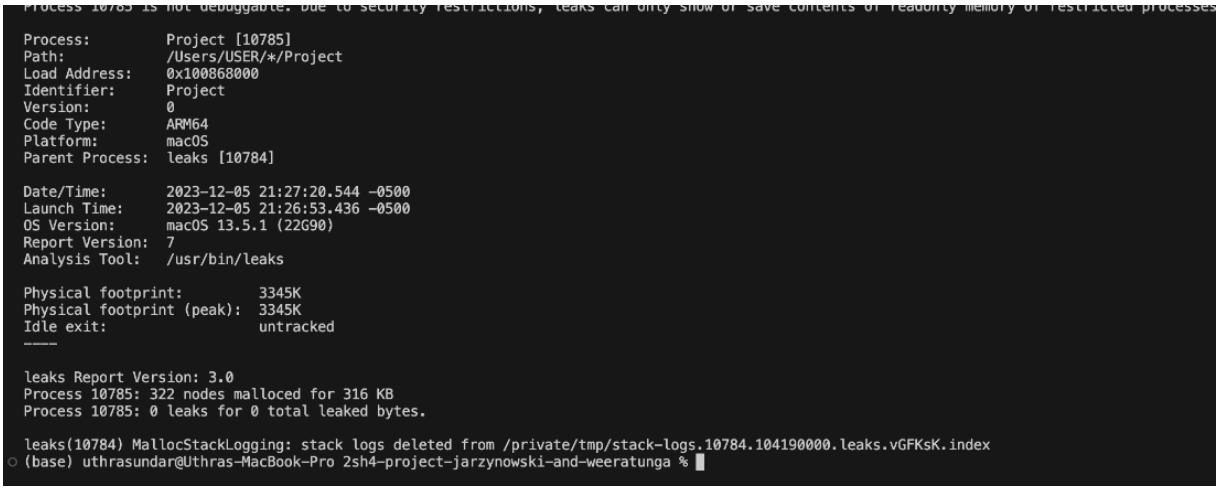
1. **[6 marks]** Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just a technical user feedback)

   The snake game offers a smooth bug-free playing experience for windows users, however not for Mac users due to the use of different printing methods in the main program; MacUILib_printf,

printf and cout. The evaluating team had to manually change the cout and printf statements to MacUILib_printf statements. To avoid possible bugs, the programmers should be consistent with their programming choices so to provide a smooth-running program compatible with different OS platform users. Moreover, due to the implementation of snake array, many if and else statements were used in the drawScreen which valid however, the else statements can be eliminated since there are no else if statement, resulting in a faster compilation due to reduced line of code for execution. For example, the for loop to print the snake could have been called first, followed by a if statement to check if the body was printed. If the body was printed, printing the borders and food elements could have been skipped by using continue inside the for. In general, avoiding the use of else statements if only checking for two conditions can lead to a more organized and readable code with faster execution time. Another suggestion to improve the gaming experience is to use a separate key to quit the game. The team did set up a message if the user quits the game but never implemented it in the run logic method. This can be fixed by adding an if statement in the run logic to check if a specific key was pressed and set only the exit flag true.

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.

   The Snake Game does not exhibit any memory leaks based on the memory profiling report. The memory profiling report indicates stable memory usage throughout the game's execution, without any memory leaks. This indicates that allocated memory on the heap, is properly deallocated within the classes and the main program preventing memory leakage, increased memory consumption and any unpredictable behaviour.



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 a great project experience working with a partner. The workload was evenly distributed among us which resulted a manageable workload along with other courses. There were a lot of debugging sessions when merging two separate codes for a single iteration. Iteration 3 onwards was challenging as the arrays were also implemented using OOD. This collaborated software development experience gave us an insight of software development procedures, exposing to higher level concepts and practices. Understanding and combining different coding styles and approaches from two team members initially posed challenges and required extra time for familiarization. Overall, the project required time, tons of debugging and thought process to yield a smooth-running game.