COMPENG 2SH4 Project – Peer Evaluation

Your Team Members _Shelby and Maryian

Team Members Evaluated _TA-Salehi___

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

Note: Our evaluated team has not completed the project, code is the same as the initial git repository given to us at the start. We will be using our code as a reference to answer the following questions.

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.

The naming of the functions corresponds with their function and output. It also includes all the function prototypes which tell us what arguments a function will take, and its input too. For example, if it is a void function, there will be no return. The preprocessor directives also provide insight into what the given header file and its functions will need to interact with (for example, if there is time.h, elements from that library like srand may be implemented, and any other header files included suggests the given functions would interact with the functions in the mentioned header files).

Positive features are that it is succinct, all features are grouped together in related categories and spaces are used accordingly to separate and improve organization. The function prototyping allows for a general summary of all the functions in each object. It is also beneficial to have prototyping in the header file and not the full implementation to separate the two – for example if a programmer is working with a client and does not want to share all the private variables in the header file.

Some negative features would be the naming of the variables in the header file, and how they are private to the class. Because we do not have the implementation of the functions and the use of the variables, the variable names are quite general and not always intuitive. For example, in our Food.h, we have variables int randx and int randy (which are used for random number generation for food position), which are not named to clearly represent their use to the user.

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.

Our evaluated team has not completed the project, so we will be using our code as a reference to answer this question. We can easily interpret how the objects will interact with each other as we first declare pointers to the classes we will be using in the global scope. The names assigned are also representative of what class they are from and their function in the game (for example, pointer to player class is named Snake because it is a snake game). In initialization, the create new objects of the classes and we can clearly see that snake takes the game mechanics and food class objects as arguments, meaning it will directly interact with the functions of those two classes. Furthermore, there are supporting comments along with the code to provide organization and clarification.

Positive features are the naming and organization of the code, with blank spaces and new lines inserted where appropriate and supporting comments to further explain any given line. For negative feature, we could have represented the interaction between food object and player object better (food generation requires the player body position to not generate food on the player's body), because it is only evident when you read line by line in the draw screen function (lines 92-101 in Project.cpp).

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.

Pros of C++ OOD vs C procedural design

- Has specifiers like private, public, protected – C procedural does not
- More secure to hide data (private scope)
- Allows for encapsulation, inheritance, polymorphism
- Pass by reference (not pass by pointer as in C)

Cons of C++ OOD vs C procedural design

- Pointers in C++ take up a lot more memory
- OOD less reusable, but functions in C procedural design can be reused in the program again and again
- Can be more concise with code especially when only a few functions are needed, sometimes OOP leads to repetitive code that is not always necessary

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.

Our evaluated team has not completed the project, so we will be using our code as a reference to answer this question. The code offers sufficient comments to explain codes and relevant debug messages. A negative feature would be the organization in the draw screen function, particularly the for loop from lines 92-101 in our Project.cpp. We could improve the readability by adding more comments to explain the interaction between the food class object and player class object, and how the food generation needs the player's position to create and place a new object.

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.

As mentioned previously, the header files incorporate white spaces such that code readability and organization is enhanced. The same can be said for the main program as each of the sections are clearly separated from one another. In the program, the code uses appropriate indentation in all instances where indentation would be applicable. The impact on code readability is most noted in the draw screen function in the main file or the move player function where the loops and conditional statements are indented appropriately so that they can be clearly interpreted. The newline formatting in the draw screen ensures that the score is printed on the next line rather than being adjacent to the last row of the game board, which enhances the visual aesthetics of the game.

An improvement to the code readability would be to have two whitespaces between function definitions instead of one so that it stands out more and differentiates from the single spaces used within each function. For example, in the main file rather that the function definitions appearing as one section, it would would make the separation of the functions more definite and obvious to the eyes.

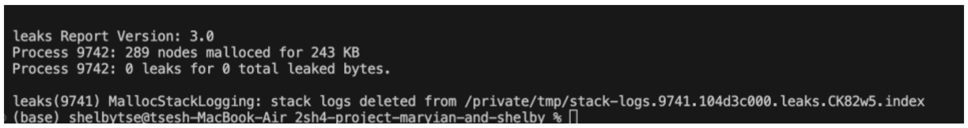
Part III: Quick Functional Evaluation

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)

Our evaluated team has not completed the project, so we will be using our code as a reference to answer this question. Yes, our snake game offers smooth, bug-free playing experience. The code exits properly on player command and game also exits when player loses, with proper exit statements displayed. No issues with compiling and no issues with segmentation fault. Snake moves smoothly and wrap around behaviour is normal.

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.

Our evaluated team has not completed the project, so we will be using our code as a reference to answer this question. In the initialize section of the main program, new was called three times and allocated memory in the heap. In the cleanup routine this memory was properly deallocated by calling delete three times as well. Therefore, as expected no memory leak found when we ran memory profiler on our code.



```
leaks Report Version: 3.0
Process 9742: 289 nodes malloced for 243 KB
Process 9742: 0 leaks for 0 total leaked bytes.

leaks(9741) MallocStackLogging: stack logs deleted from /private/tmp/stack-logs.9741.104d3c000.leaks.CK82w5.index
(base) shelbytse@tsesh-MacBook-Air 2sh4-project-maryian-and-shelby % 
```

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 helpful to collaborate because having another team member was useful for debugging and locating errors and developing code. It was difficult when collaborating on iterations where both partners were needed. For iteration 3, we only worked on one partner's computer because it would be inefficient to switch and push/pull the code each time.