COMPENG 2SH4 Project – Peer Evaluation

Your Team Members _Darren_Temporale_ _Konstantin_Delemen__

Team Members Evaluated _Anna_Dienstmann_ _Remsha_Akbar_Hussian_

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

Part I: OOD Quality

1. **[6 marks]** OOD is about sensible code modularisation. 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 header files of each object are well organized and have clear comments that explain the purpose and functionality of each class, its data members and its subsequent member functions. The header files make use of PascalCase for the classes and camelCase for the data members and member functions. This enhances readability and helps avoid confusion. The header files have made proper use of private and public access specifiers as well as good use of const qualifiers to avoid accidental changes. Lastly, the header files use destructors as well as copy constructors which are a good practice for avoiding memory leaks

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.

   The main program loop is easy to interpret and understand as it follows the basic four steps of get input, draw screen and loop delay. The main program loop has some comments to explain the purpose of each function as well it uses descriptive function and variable names. Some things that can be improved include making separate classes or objects to represent the game entities. Examples of this include the board, snake and food. Instead in the draw screen function, it uses a matrix and a pointer to objPosArrayList as well as objPos object. As a result, the code is less modular and prone to errors as the data and the logic have not been encapsulated and abstracted.

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.

   a. Pros:
      i. Much cleaner and easy-to-read main program loop
      ii. Changing core functionality can be easier (with a good OOD implementation)
      iii. It is significantly easier to reuse code for different applications

    iv. Encapsulation hides the implementation details of the data and the code that it is executing. Specifiers such as private, public and protected are not available in C

b. Cons:
    i. Tracking down a bug/specific line of code can be more difficult (especially if it's buried many classes deep)
    ii. In some instances, C procedural design may have an advantage over C++ OOD in terms of performance as it avoids the overhead of managing memory allocation.

Part II: Code Quality

1. **[5 marks]** Does the code offer sufficient comments, or deploy 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.

    a. There are plenty of comments and explanations to aid in code understanding throughout most of the code. One area that stood out as a spot that could use a little more explanation is the generateFood method – things like explaining why x+1 and y+1 are used, along with which blockOff is intended to be evaluated in the while loop condition. Overall, the comments are great, it's just that method that was a little confusing and hard to parse.

2. **[4 marks]** Does the code follow good indentation, add sensible white spaces, and deploy newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.

    a. When printing the move count and the score beneath the game board, it would be easier to read at a glance if they were printed on separate lines or separated in some way other than a space.

    b. The code is indented well but could use empty spaces and blank lines to improve readability - particularly in the player class, many spots could benefit from white space/separation.

    c. The code could be made to be more consistent in the use of spaces between operators, parentheses and commas. Throughout the code, there are inconsistencies in the way spaces are used. For example in updatePlayerDir inside of player class, it does not use spaces between any operators or parentheses. This makes it harder to read and intern hinders code readability

Part III: Quick Functional Evaluation

1. **[8 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 technical user feedback)

   a. When moving in any direction (UP for example) if you hold the input key to move in the opposite direction ('s' for the example), you will eventually move in that direction (it takes about 1 second of holding 's' to move down). If this is done with a snake length of more than 2 units/chars, this will end the game due to the player's head being in the same position as the body parts. One way to fix this bug could be to look at the update player method and consider ways to merge the "checking if the input is illegal with direction" section into the switch case that changes myDir. Having a system that prevents the direction from changing based on the current direction (instead of the direction <u>and</u> the input) could fix this bug.

2. **[6 marks]** Does the Snake Game cause memory leaks? If yes, provide a digest of the memory profiling report and identify the possible root cause of the memory leakage.

   a. No Memory leak was found upon running a DrMemory scan, all uninitialized reads/writes came from MacUI_Lib or mingw functions, indicating proper memory management practices.
   b. There is no memory leak present in the code since everything allocated on the heap has a matching deallocation either in the clean-up function or in a destructor. As a result, there is no memory leak.

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.

Our experience in our first collaborated software development through this project was very positive and rewarding. We efficiently divided the tasks according to our strengths and preferences and communicated frequently throughout the development. Some of the things that went well included making a timeline to reach milestones in the project, using object-oriented design principles to structure our code, testing and debugging our code regularly and giving and receiving constructive feedback to each other. Some of the things that could have been improved include dealing with compatibility issues as we used macOS and Windows for development and having more foresight for memory leaks during the development of the code.