COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members ahmem121_____ yangj351_____

Team Members Evaluated jiangr42_____ li3081_____

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **30 marks**. Do not exceed 2 paragraphs per question.

Peer Code Review: OOD Quality

1. **[3 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 well-organized with functions handling specific tasks: GetInput() for user input, RunLogic() for game updates, DrawScreen() for creating the board and simulating the game, and LoopDelay() for controlling the speed. The GameMechs class controls the core game mechanics, while the Player class tracks the snake's movement and body, updating its direction and position based on user input. The snake's body is managed using the objPosArrayList, which adds new segments to the head and removes the tail. The FoodBin class handles food generation and interacts with the Player to grow the snake when food is eaten. Overall, the interaction between objects is clear and well-structured, making the game logic easy to follow.

However, there are some areas for improvement. Currently, only the food that is eaten by the snake is randomly spawned, but it could add more to the game if all food items (not just the one eaten) were randomly generated. Furthermore, the winFlag in the game is a bit unusual for a snake game. In most snake games, the focus is on avoiding collisions and growing the snake, rather than achieving a specific win condition. It might be worth reconsidering the need for a winFlag altogether or implementing another feature in a way that aligns better with a typical snake game.

2. **[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

- **Modularity**: By breaking down our code into separate components, it was easy for both team members to work on different parts of the project at the same time.
- **Improved code organization and maintenance:** the snake project uses many classes implemented in different files — the organization was logical and made it easier to access, analyse, and edit different parts of the code.
- **Reusability:** We were able to reuse code through inheritance, making the game development process more efficient — the Food, GameMechs, and Player classes inherited many of the objPos and objPosArrayList classes' characteristics. Those were then used to create specialized functions specific to those classes in less time

- **Abstraction**: By building upon the objPos class, we were able to focus on the larger picture and the logistics behind adding more complex details such as moving the snake around the board by inserting/removing units instead of individual action lines.
- **Error Prevention:** Easier to isolate errors to specific parts of the code (e.g. classes) and debug within a smaller scope.
- **Complexity:** The implementations of some functions for simple tasks may have been unnecessarily complex.
- **Memory Leakage:** The frequent use of pointers across different files meant it was easy for memory leakage to occur if dynamic memory allocation was not handled properly.

C Procedural Design Approach

- **Simple implmenetation:** more straightforward, which may be easier to implement for smaller projects like the PPAs. As a result, may also require less resources!
- **Difficulties on a larger scale:** The snake project was a lot more complex than the PPAs, and the project became harder to manage as a result — if procedural programming was used, it would be very hard to organize and maintain.
- **Unnecessary code Duplication:** Code is not as reusable as in OOD, so code may be repetitive.
- **Poor Maintainability:** When applying changes to tne part of the code, major changes may have to be made in other parts as well.

Peer Code Review: Code Quality

1. **[3 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 code offers sufficient comments, and the explanation about the coding is generally clear, making it easy to understand how the code functions. The function names are descriptive, and key sections, such as the food-eating logic and snake movement, are explained well. However, in functions like DrawScreen(), where multiple other functions are called (such as clearBoard(), addSnake(), and addFood()), adding brief comments explaining why these functions are being called in the function would improve understanding. While the overall comments are clear and helpful, providing a little more context in these function calls would help with understanding for someone reading the code for the first time. Overall, the comments are suffixien and contribute to the code's readability.

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.

Yes, the code follows good indentation, spacing, and newline formatting. Single white spaces in expressions improve the readability of the code, such as in assignment statements like "eat = false;" in the FoodBin.cpp file. Newline formatting at the end of each statement and double newline formatting in between functions allow the reader to differentiate sections of code with different purposes and follow the code's logic. Blocks of code that are a part of a parent structure, such as loops, conditionals, or functions are consistently idented to show that they are a part of the structure. In larger parent structures, lines with similar purposes are indented at similar levels. For instance, for the updatePlayerDir(char input) function (player.cpp file), the switch keyword under the

if statement is indented, each “case” keyword under the switch statement is indented at the same level, and the actions to be executed for each case are indented to the same level. The only noticeable place where this was not followed is in the eatFood(objPos food) function (FoodBin.cpp file) — though it does not noticeably hinder the readability of the code, the indentation convention for switch statements described above could be followed.

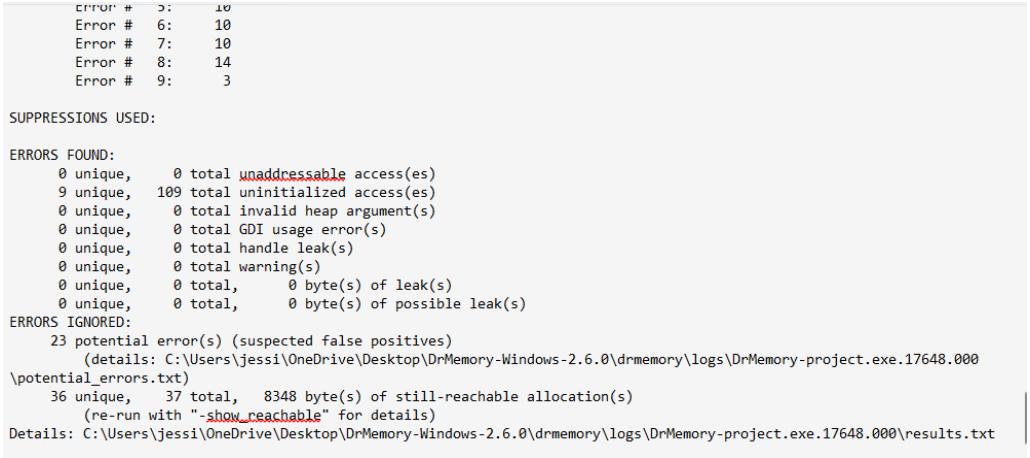
Peer Code Review: Quick Functional Evaluation

1. **[3 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 technical user feedback)

Based on the provided code, the Snake Game appears to offer a smooth, bug-free playing experience. The game logic for movement, food generation, and collision detection is well-structured, and there are no immediate issues that would affect gameplay. The game mechanics, such as snake growth and food consumption functions work as well as expected.

2. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.

No, as seen in the image of memory profiling report below, the Snake Game does not cause memory leak - this is indicated by the lines under “ERRORS FOUND:,” “0 byte(s) of leak(s)” and “0 byte(s) of possible leak(s).” What is observed in the report aligns with the correct implementation of the objPos and objPosArrayList destructors in the respective files — these destructors handle dynamic memory allocation by freeing up memory allocated to the structures in the constructors.



```
        Error #   5:     10
        Error #   6:     10
        Error #   7:     10
        Error #   8:     14
        Error #   9:      3

SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      9 unique,   109 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:
     23 potential error(s) (suspected false positives)
         (details: C:\Users\jessi\OneDrive\Desktop\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-project.exe.17648.000
\potential_errors.txt)
     36 unique,    37 total,   8348 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
Details: C:\Users\jessi\OneDrive\Desktop\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-project.exe.17648.000\results.txt
```

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team’s implementation in addition to yours, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

The design of the objPos class, which uses the Pos struct to store the x-y coordinates, is a good choice because it keeps things organized. By separating the position from the symbol, it makes it easier to manage and update the position data, especially since the snake's position changes frequently.

However, using a pointer for the Pos struct is very difficult, as you need to make sure you would deal with memory management correctly in order to avoid memory leaks. Even though this adds some extra work, it's a flexible design that allows for easier changes or updates to the position data if needed. Overall, the design makes sense as long as the memory is handled properly and no leaks occur.

2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. <u>You are expected to facilitate the discussion with UML diagram.</u>

ObjPos
+ posX: int + posY: int + symbol: char + accessible: bool
+ objPos() + objPos(poxX: int, posY: int, sym: char, accessible: bool) + setObjPos(o:objpos): void + setObjPos(poxX: int, posY: int, sym: char, accessible: bool): void + getobjPos(): objPos const + getSymbol(): char const + isPosEqual(refPos: const objPos*): bool const + getSymbolIfPosEqual(refPos: const objPos*): char const

Above is a UML objPos class diagram for a design alternative to the one used in the project. In this design, the x and y positions of the objPos class are no longer accessed using a struct pointer, but instead are individual integer members of the class. The advantage of this method is simpler memory management as these variables are included directly in the class as fields, so there is no need for dynamic memory allocation / deallocation. It also allows for simple coding structure as dereferencing the pointer is not needed to access the x and y coordinates. As well, by using this design, fewer bugs may be encountered as issues may occur with accidentally dereferencing null pointers or having dangling references. However, the disadvantage is that when our program becomes larger, using the passing pointers method may be more memory efficient to avoid copying entire structures.

Another key change to the objPos class in this design is the adding of a boolean member called accessible which can be set to true or false. By setting this value, the member can be used to "block off" specific combinations of x and y which may be useful in implementing different features of the game. For instance, when implementing the food generation: we can check for a specific combination of x and y, if accessible is true — if true indicates a player element is there, we can say for true, cannot generate food there.