COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Kristin Figueredo (figuerek) Delayna Guenther (guenthed)

Team Members Evaluated siddin25 salama20

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 **25 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 in the main function is very intuitive wherein it follows a logical consecutive sequence. First, the initialize function is called which initializes all pointers onto the heap and generates a piece of food with random coordinates. Next, the program enters a while loop which checks to see if the exitFlag has been set to false. If not, the program will repeatedly get any user input and save it to a variable under myGM (game mechanics), run the game logic where the program will process any input and update the player movement mechanics, draw the snake and food onto the screen and finally, create a time delay which will slow the process down, so it is more user friendly. If the exit flag is set to false, the program enters a clean up function where the screen is cleared and any initialized objects on the heap are freed.

   One of the positive aspects of this code are its clear structure where the team maximizes readability by using easy to understand function and variable names. Every object's name mainly corresponds to is function which improves readability. Another positive is the proper use of pointers on the heap. This group intuitively initializes the pointers in the initialize function, uses them throughout GetInput(), RunLogic() and DrawScreen() and effectively removes them off the heap in CleanUp(). This team could improve upon readability of their code by including comments explicitly explaining what each pointer is used for and how they interact with each other. For example, without looking at the Player.cpp file and the GameMechs.cpp file, I could not understand why the collectAsyncInput() required the player position. Including comments would improve the readability.

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.

   Pros

   - C++ OOD has Enhanced Organization
     - Everything that would be associated with a certain type of data, (whether it be specific corresponding values, functions or methods) is organized to be corresponded with the datatype.
   - C++ OOD has Enhanced Reusability

    - Everything can be repeatedly accessed in multiple files while in C procedural programming, a lot more copy and pasting was done.
- C++ OOD has Easier Debugging
    - If there is something wrong with a certain area of game, it is easier to debug as you can pinpoint which area is malfunctioning. In C procedural design, it is harder to locate where specifically the error lies.

Cons

- C++ OOD has a longer implementation time
    - There is a lot of building on top of older files, repeatedly making the minimum 4 functions for every class and implementing various setter, getter and manipulation functions which increases development time.
- C Procedural Design is more straightforward
    - There is a more sequential design which is easier to comprehend straight away.

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 offered sufficient comments that were detailed and made it easy to follow along and understand their logic. In every class file they had code explaining each function implemented and explaining what they did throughout each function. They could have added more comments throughout the main project file, as anyone that does not know this project well may have a difficult time understanding the code. Also, they left some commented out code in the files that could have been deleted for ease of reading.

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 good indentation, and everything is organized in a way that is easy to understand. They put a good amount of white space which makes it easy to follow where one function ends, and another begins. The white space between variables makes it easier to read the values and the arithmetic equations of variables. However, in some of their functions, there is a lot of white space after the lines in the functions. To fix this I would delete some extra blank lines to shorten the overall code.
   Also, the printed statements all begin on a new line and make sense as they print the game's score, the current coordinates of the head of the snake, and an explanation for pressing 'esc' to exit the game. These printed statements make it easier to play the game with ease.

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)

Yes, this snake game offers a very smooth and bug free experience. The team could have made the overall game design easier to understand for a new user by printing out controls/game instructions and an end game message, but the game itself showed no issue. The wrap around, snake elongation, random food generation and scoring mechanics all functioned correctly. Instead of putting the 'game over message' in the GameMechs.cpp file, I would have put it in the clean up file so the user could view the message after they have lost, (instead of just for a second before the program terminates).

In terms of code reusability, the generateFood() function uses hardcoded numbers for the generation of food coordinates. This reduces the reusability of this function as it would not work in any other grid/gameboard size. Instead, I would use a getter method from the GameMechs class to get the current gameboard size to ensure I can use this function for all gameboard sizes.

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, this game does not cause any memory leak. Whenever this team used heap space, it was freed in a destructor or clean up function.

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, 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 object design of objPos is unsensible because of the presence of typedef struct. Typedef structs are normally used in C and are unnecessary in C++. Also, structs in general can be avoided in C++ as they are essentially the same thing as a class. Classes can provide the same functionality and accessibility as structs and since we already have a class called objPos there is no need to have an additional struct inside. Also, removing the struct will make calling the class easier because we will no longer need to add '.pos->x' to access the x variable; instead '->x' is sufficient. Additionally, dynamic memory needs to be allocated in the struct which can lead to memory leaks, this can be avoided by using the class instead.

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(s).</u>
   An alternative objPos class design could be to remove the struct and instead initialize the variable x and y inside objPos and create a function to access them like the way symbol is initialized and getSymbol is called.
   For example, the new UML diagram would be:

objPos
- symbol: **char** - x: **int** - y: **int**
+ objPos()

+ objPos(xPos:**int**, yPos:**int,** sym:**char**)

+ setObjPos(o:objPos): **void**
+ setObjPos(xPos: **int**, yPos:**int**, sym:**char**): **void**
+ ~objPos();
+ objPos(Pos1: **objPos& const**)
+ operator=(Pos1: **objPos**): **objPos**
+ getObjPos(): **objPos const**
+ getSymbol(): **char chonst**
+ isPosEqual(refPos:**const** objPos*): **bool const**
+ getSymbolIfPosEqual(refPos:**const** objPos*): **char const**

+ getX(): **int const**
+ getY(): **int const**

This new UML diagram removes the need for the struct and will still function in the same way. Private integers x and y were added along with getX() and getY() functions to make it easy to access the x and y values separately. Also, because we no longer need to have Pos* pos we don't need to delete any memory in the deconstruct function. Overall removing the struct simplifies the code.