COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Amrit Virk (virka12) Pranav Kukreja (kukrejap)

Team Members Evaluated Team name: aaaaaa, Zhuk50, Khans294

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.

   After examining the main logic within the main loop found in project.cpp, we've determined that everything can be easily interpreted and understood just by reading it over. We found that the team made good use of comments as the more complex lines had brief comments explaining their functionality, making it easier to understand. Moreover, the team also chose appropriate variable names, further improving the readability.

   One thing that we found interesting was the way that this team set up their logic for printing the game elements onto the board. The logic that they decided to use is different from the way we implemented it. Their code takes provides a more efficient method for achieving the same result.

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 of C++ OOD approach:

- Breaks workflow up into modular pieces which can be easily edited and altered
- Simplifies the code when working with larger projects
- Due to everything being divided into various modules, expanding the code becomes much easier as either new classes can be made to expand the code or adding content into pre-existing classes
- The OOD design can also help increase the readability of the code as code can be condensed and stored in one central spot, making it easier to find specific variables.

Cons of C++ OOD approach:

- Learning and adapting to the OOD approach can be challenging at first, especially if coming from procedural design approach
- Can over complicate smaller projects by introducing unnecessary classes and steps whereas using procedural would make everything easier
- Can create a lot of clutter and confusion if not implemented correctly

- Can require a lot of back and forth to understand the code

Pros of C procedural design:

- Makes code and process straightforward as everything is done step by step, making it easier to read and understand
- Mistakes in logic and syntax are easier to find and fix since everything is written in order so it's easy to find where the code is going wrong
- Ideal for smaller projects

Cons of C procedural design:

- As the project gets larger, updating and increasing the code can be difficult at times since you'd have to go through the entire code to find the specific section you want to work on and expand it from there.
- Can create confusion in variable management as you can have duplicate variables depending on where they are declared and stored

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.

   As mentioned from before, the code has sufficient comments in places where the code becomes more complex alongside appropriate variable names, which all aid in enhancing the readability and quality of the code. If I had to improve one thing, I would add comments in other files as well such as food.cpp but they since those files aren't too code heavy, they can do without them.

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 formatting of the code is great as it follows all the criteria to make a visually appealing code. All the lines are properly indented whilst leaving adequate white space between blocks of code to improve readability.

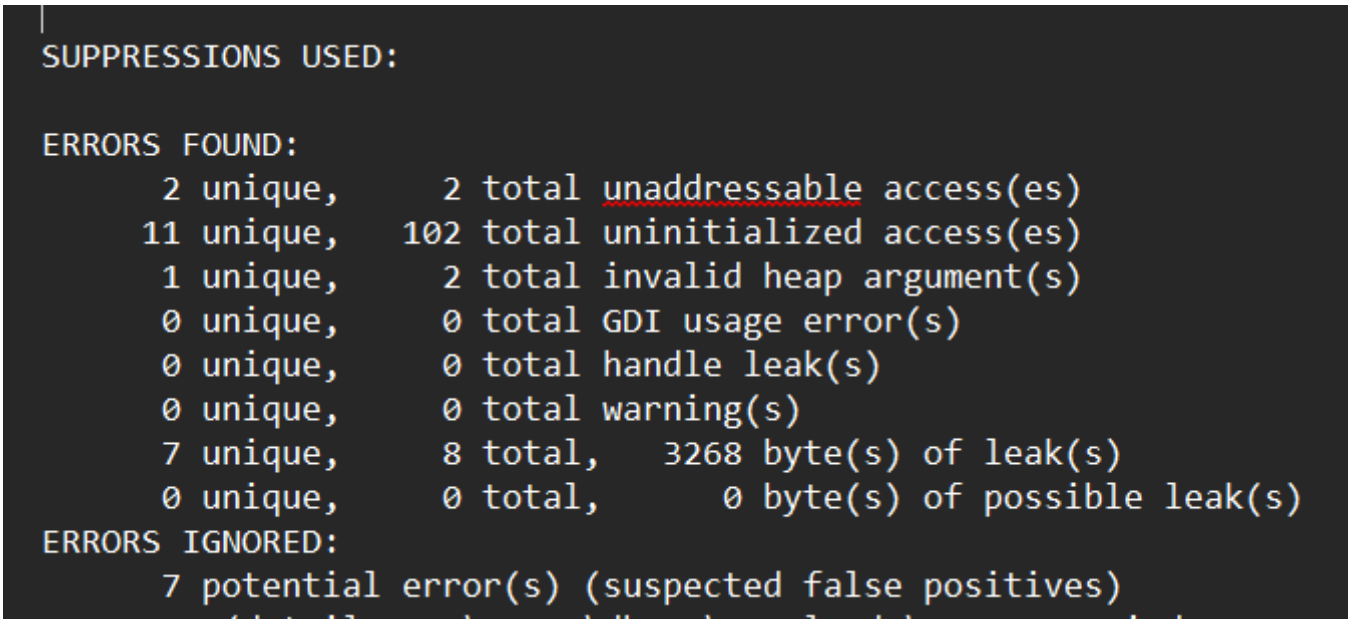
Peer Code Review: Quick Functional Evaluation

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)

   Although the gameplay is smooth and bug free, unfortunately the game over screen is bugged out. When the snake hits its own tail, rather than the game ending nicely, a malloc error appears indicating that a pointer which isn't allocated is being freed. From our understanding, the root cause would be a freeing of pointer which wasn't defined meaning either an unneeded delete line was added or a pointer was forgotten to be defined. Moreover, the manual game ending key bind was also not working so we had to force the game to end by either using the force kill command or by hitting the snake into itself.

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.

The snake game does have a decent amount of memory leakage.



One potential spot for memory leakage can be within the player destructor in the Player.cpp file on line 22, as within the constructor different syntax is used in comparison to the destructor.

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?

We do not think that the compound object design of the objpos class is sensible as it creates unnecessary lines of code making the code look more complicated and confusing than it really is. Instead, an easier and more code efficient method would be removing the extra typedef struct class all together and moving its contents within the objpos class itself. This creates a better solution our opinion as although in theory having them separated is better for the code, the contents of the Pos struct aren't complex enough to warrant another class. If there were more variables within the class then separating them would be ideal but in this case it simply doesn't make sense.

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. You are expected to facilitate the discussion with UML diagram(s).

As mentioned above, the design can be improved by removing the redundant pos struct class and moving its contents, being x and y location variables, into the objpos class itself. These variables would be stored as private data members within the objpos class. As a result of this getter and setter methods would need to be involved to interact with the variables.

UML Diagram:

objPos
-x: int -y: int -symbol: char
+ objPos() : void + objPos(xPos : int, yPos : int, sym : char) : void + setObjPos(o : objPos) : void + setObjPos(xPos : int, yPos : int, sym : char) : void + getObjPos() : objPos + getSymbol() : char + getSymbolIfPosEqual(refPos : const objPos*) : char + isPosEqual(refPos : const objPos*) : bool