COMPENG 2SH4 Project - Peer Evaluation [30 Marks]

Your Team Members: arora_net (meloi3, yingj14)

Team Members Evaluated: jaina78-sirisenp (jaina7, sirisenp)

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, making it easy to understand how the game processes input, updates logic, and draws on the screen. The use of objects like Player and GameMechs helps keep the code clean and focused on specific tasks.

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 PP3A.

   Pros:

   - Organization: putting the underlying logic within classes allows the main program loop to look a lot more readable
   - Portability and modularisation: the OOD approach allows you to build the game piece by piece and allows for easy implementation with other classes

   Cons:

   - While the OOD approach allows for a very clean main program, it is also very vague. The main logic of how the snake game operates is hidden within the methods of several other classes which can be annoying when first trying to understand the codebase.
   - Implementing a change can sometimes be very tedious. In the case of changing the food class to accept an ArrayList instead of solely an objpos, there were a lot of underlying methods that needed to be refactored in order for this change to work

---

Peer Code Review: Code Quality

1. **[3 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 shortcomings are observed, discuss how you would improve it.

   The code is well documented with clear comments that make it easy to follow and understand the functionality of each section and method. Everything is explained effectively, making sure the intent behind each part of the code is clear to the reader.

2. **[3 marks]** Does the code follow good indentation, add sensible white spaces, and deploy newline formatting for better readability? If any shortcomings are observed, discuss how you would improve it.

   The code follows good indentation, uses sensible white spaces, and applies newline formatting effectively, making it very readable and easy to navigate. The formatting is consistent throughout, and no issues were observed.

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.)*

   The program does not exhibit any sort of buggy features and runs very smoothly. The only change we would make would be to increase the boarder size so the player has more room to move around.

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

   The program did not contain any memory leaks.

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?

   This method was sensible if ease of use was the only factor being considered. Using the struct allowed us to easily access the player's position simply through the arrowhead operator(->). Had the player x and y position been a member of the objpos class we would have had to use getter methods or instantiate a new object to get the current position. In terms of following the

principles of OOD design, using this struct doesn't follow proper encapsulation techniques of hiding member variables in getter and setter methods.

2. **[4 marks]** If yes, discuss an alternative objPos class design that you believe is relatively counterintuitive compared to the one in this project. If no, explain how you would improve the object design.
   You are expected to facilitate the discussion with a UML diagram.

Opposed to using a struct to store the player objects x and y position, member variables alongside getter and setter methods can be used to mimic the same behavior while also allowing for encapsulation. More specifically, member variables playerX and playerY will store the x and y positions of the player and can be modified through the getPlayerX, getPlayerY, setPlayerX, and, setPlayerY methods.

objPos
+ playerX: int + PlayerY: int + symbol: char
+ objPos() + objPos(xPos: int, yPos: int, sym: char) + objPos(const objPos &obj) + ~objPos() + setObjPos(const objPos &o): void + setObjPos(xPos: int, yPos: int, sym: char): void + getObjPos(): objPos + getSymbol(): char + isPosEqual(refPos: const objPos*): bool const + getSymbolIfPosEqual(refPos: const objPos*): char const + getPlayerX(): int + getPlayerY(): int + setPlayerX(): void + setPlayerY(): void

*New methods and member variables are in red