COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members nivarths, suresl5 (Sivansh and Lhavanjan)

Team Members Evaluated baikh1, palumj3 (WeWill12)

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 demonstrates a well-structured design, effectively separating input handling, game logic, rendering, and delay into distinct functions. This clear organization adheres to best practices, making the code easy to follow and maintain. The use of object-oriented principles is commendable, as objects like `Player`, `GameMechs`, and `food` each have well-defined responsibilities that interact cohesively to drive the game's mechanics. The code handles dynamic memory allocation properly, ensuring resources are freed when the game ends, and the logic for movement, collision detection, and score management highlights thoughtful design. However, while the program is well-organized, the RunLogic function could benefit from further abstraction to simplify its flow and make it more intuitive. For example, the logic for handling food consumption, generating new food positions, and incrementing the score could be encapsulated within a method in the food class, such as food->handleConsumption(playerPosList). This would centralize food-related operations, making the code cleaner and aligning it better with object-oriented principles. This refinement would make the game logic easier to read, maintain, and extend. Overall, the program reflects a strong foundation, with minor refinements to improve encapsulation and maintainability further.

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.

The C++ object-oriented design (OOD) approach in the project has clear advantages over the C procedural design in PPA3, especially for larger, more complex programs. By using classes like `Player`, `GameMechs`, and `food`, the C++ code keeps data and behavior neatly organized, making it easier to manage, extend, and reuse. It also makes adding new features, like obstacles or advanced mechanics, much simpler because each class handles its own responsibilities. On the flip side, OOD can feel a bit more challenging to learn, and dynamic memory management adds some complexity and overhead. In comparison, the procedural approach in C is more straightforward and faster to write, with direct control over the program flow. However, it relies heavily on global variables and tightly coupled logic, which can make the code harder to debug and expand. While C works well for smaller programs, the OOD approach is a better fit when you're building something bigger and want it to be clean, flexible, and future-proof.

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 demonstrates a solid foundation with clear class definitions and well-structured logic. The comments provide a good overview of the overall functionality. However, by adding a few more explanatory comments to specific sections, we can elevate the code's clarity and maintainability to new heights. In particular, pointer usage and intricate data structures could benefit from these additional comments. By applying additional comments, we would not only enhance the code's readability for current developers but also serve as valuable documentation for future contributors who may need to understand and modify the code. By taking the time to write clear, concise, and informative comments, we can create a code that is not only functional but also a pleasure to work with. This investment in documentation will aid you in the long run, reducing the time and effort required to understand, maintain, and extend the code.

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 demonstrates an exceptional level of formatting, adhering to best practices with consistent indentation, appropriate use of whitespace, and proper newline formatting. This attention to detail significantly enhances readability, making the code effortless to understand and maintain. The inclusion of clear and concise comments further elevates comprehension. By following these guidelines, the code is not only functional but also is not a hassle to work with, streamlining development efforts, promoting long-term maintainability, and overall having a positive coding experience.

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 game runs smoothly with and offers a bug-free playing experience. There are no buggy features that hinder the playing experience of the game.

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 game does not cause a memory leakage based on the drmemory result file showing 0 bytes of memory leak. This was also handled well in the cleanup routine, where the team deleted and deallocated any memory allocated for the members constructed on the heap.

Project Reflection

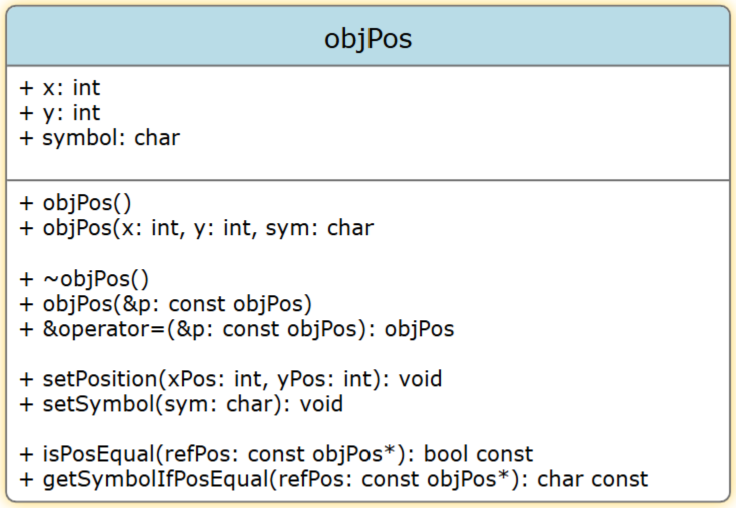
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

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

The compound object design of the objPos class is unusual, but it has its benefits and drawbacks. In terms of benefits, the encapsulation allows for easy coordinate data manipulation, making the logic reusable in various contexts. It also allows for flexible heap allocation, which might be useful in situations that require coordinates to be frequently reallocated. Finally, this encapsulation ensures that all the objects in the game use the same representation of position and symbol, allowing simpler interactions. However, this design does have its flaws. Introducing the Pos struct as a dynamically allocated pointer considering it seems simpler to manage the two integers, x and y, instead of unnecessary memory allocation. Additionally, this design increases risk of dangling pointers, memory leakage, and buggy behaviour due to incorrect deletion/initialization/usage of the pos pointer. Finally, using the heap memory with every instance of the objPos class may cause a decline in performance due to constant allocation/deallocation, which can hinder game performance. Based on this, I think the compound object design of objPos class is not sensible.

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>

Instead of the compound design, x and y can be directly embedded as members of the objPos class, eliminating the need for heap allocation through the struct. The updated UML diagram would look like this:


objPos
+ x: int
+ y: int
+ symbol: char
+ objPos()
+ objPos(x: int, y: int, sym: char
+ ~objPos()
+ objPos(&p: const objPos)
+ &operator=(&p: const objPos): objPos
+ setPosition(xPos: int, yPos: int): void
+ setSymbol(sym: char): void
+ isPosEqual(refPos: const objPos*): bool const
+ getSymbolIfPosEqual(refPos: const objPos*): char const

By integrating x and y as integer data members in the objPos class, the design is simpler, which will make it easier for developers to understand and use. It also eliminates dynamic memory allocation, which will reduce the chance of memory leakage, dangling pointers and errors. Finally, it is more efficient with a better representation of position and symbol, allowing for smoother game performance, due to the elimination of constant memory allocation and deallocation. This is how I'd improve the design, through the removal of the struct and making x and y data members.