COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members yuw79_____ bandaa2_____

Team Members Evaluated anandd4_____ dhodiv_____

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.
   Yes, from the initialization we can clearly see GameMechs is first created and serves as a member object in the Player class. The Player class is then used as a member object of the food class. But the current_Player being initialized twice might lead to unexpected behavior. Also, circular dependency between Player and Food could potentially complicate object management.

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:

   Combine data and functions in objects, improving modularity

   Encapsulation: Provide data access control features, improving data security

   Encourage reusable components through inheritance and polymorphism

   Parallel development. Each team member can work on their own class individually without affecting the other team member's code.

   Cons of C++ OOD:

   Increase the complexity. Can be difficult to design and define relationships between objects.

   Need careful handling of memory

   Lots of boiler plate code needed to create a simple object class. Not viable for smaller programs.

   Pros of C Procedural:

   Easier to understand and implement. Can also be more flexible than OOD which is a more rigid design.

   Faster execution for smaller programs

   Developers have full control over data and logic

   Cons of C Procedural:

Harder to manage as the project grows

Heavily rely on global variables, which can lead to unintended side effects

Poor code reusability

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 includes sufficient comments in most parts, particularly regarding the general purpose of functions. Additionally, variable and function names are meaningful and enhance readability. However, it lacks comments on certain key algorithms, such as the player position warping and the finite state machine logic, which could benefit from more detailed explanations.
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 demonstrates good formatting in terms of indentation, whitespace, and newlines. However, improvements can be made in separating logical blocks within loops, switch statements, and dense functions for better readability and maintainability.

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 delivers a good overall playing experience without noticeable bugs. My only suggestion would be to enhance the game instructions, such as adding a special exit message when the player manually chooses to quit the game since in both cases when the player loses or exits the output is 'game over'.
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 program effectively prevents memory leaks by properly handling heap-allocated data in both the destructors and the main program.

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?
   No, allocating memory for Pos on the heap in the constructor adds unnecessary complexity and overhead. Using a struct for Pos is also a little strange since structs aren't typically used in C++. Perhaps making data members like XPos and YPos or having an array with two elements representing the XPos and YPos could also make it simpler.

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).
   A simpler approach could be to use a member variable of type Pos directly instead of a pointer. It can be represented by a UML diagram: