COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members Philip Lee Devin Gao

Team Members Evaluated Huangr86 phama21

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.

   After examining the main logic in the main program loop, it is relatively easy to interpret how the objects interact with each other. In the RunLogic function, I can clearly observe that the Player class interacts with Food and GameMechs for collision check, update movement and the score system. Objects interaction can also be seen in food generation, the code, food->generateFood(player->getPlayerPos());, ensures that the new food does not appear where the player exists and does so, by obtaining the player position from the player object. One thing that would be beneficial, while not exactly related to the main logic code, is to add a bit more commenting since the comments only explain the feature in simplest terms. If a beginner were to look at the program, the code would be confusing so some commenting on the process explained above would be beneficial for more clarity.

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++ uses classes and objects to group related code, making the code easier to read and organize.
   - Object-oriented design allows for code to be reused across different projects.
   - Easier to expand and add new features without breaking existing functionality.
   - Multiple people can work on separate classes or functions independently, easier to collaborate.

   Cons:
   - C++ OOD approach can take more time due to its structure and extra features.
   - For simpler and smaller programs (like PPA3), OOD can seem excessive compared to the straightforwardness of procedural design

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 provides a reasonable number of comments. These comments help describe the purpose of functions, variables, and specific logic. Majority of the code is easy to understand with the help of the descriptions. However, some areas could use a little more explanation. For example, the Food class has little to nothing comments at all. To improve understanding, comments could clarify how the generateFood function works and how it interacts with other classes. A more thorough documentation style, like commenting on the role of each function, would greatly help.

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 consistent indentation, newline formatting and sensible white spaces throughout the project. Nested loops and conditions are aligned and separated correctly, making it easy to follow the logic. Use of white space in separating different sections help keep the code to a clean format. The only part where more white space could be beneficial, is in the DrawScreen function. Additional white space or blank lines to help separate the different cases will increase its 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)

   The game generally provides a smooth, bug-free experience. However, there are some areas for improvement. Firstly, the prompt says that '@' will increase the score by 10 and the snake size by 1. But it only increases the score and not the size. I think this is just simply a typo, as '@' and '&' were confused and flipped. Another buggy feature is the wrap-around logic. The snake should reappear on the opposite side of the screen when it moves past a border. However, this logic is flawed because the snake can move along the border without wrapping immediately. This suggests that the position-checking or wrap-around implementation is incomplete. Debugging recommendation would be to add boundary-specific unit tests to verify the wrap-around functionality and simulate edge cases, such as placing the snake head on the border or generating food near the snake body.

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 any memory leak, and all allocated memory were deleted at the end of the program. The code shows proper dynamic memory management by using new for allocation and delete during cleanup.

Project Reflection

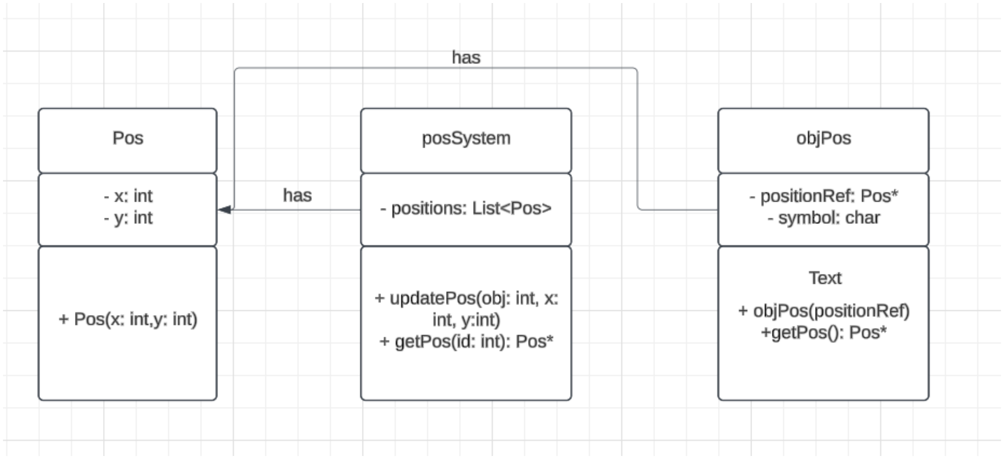
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to yours, reflect on the following questions:

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

   After reviewing the other team's implementation of the Pos struct in addition to ours, the compound design of objPos is sensible. Since the Pos struct only contributes to the positioning of the object while the objPos class handles more complex object-related attributes, it offers better organization and reusability in the code. In addition, if the project were to go even more above-and-beyond with complex features, having a Pos struct allows better ease of extending. For example, if the project introduced 3D coordinates instead of 2D, having a Pos struct to solely extend the coordinate system is much more efficient than changing the entire objPos class without the struct. Lastly, having a Pos struct grants the user ease of passing the coordinates around the code, and can be managed independently. This is especially useful since the project code works the objPos class in different contexts.

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.</u>

   An alternative objPos class design that is relatively counterintuitive to the project's implementation is having the position data managed externally instead of being within the objPos class. More specifically, the position would be handled by a separate class. As shown in the UML diagram below, we can create a separate class "posSystem" managing all x and y positions related to the objPos class. With this design, the objPos class would only reference and interact with posSystem without any internal state related to the position. Pos would also be a class instead of a struct, still containing its x and y coordinates but now can be utilized through the posSystem class. posSystem includes a list of Pos objects reperesenting positions for numerous objPos instances (like the snake body) and handles the logic for moving and updating positions. The methods below are only a sample of what could be done for each class. The reason this design is counterintuitive is because it removes the association between the object and its position, making it feel less direct and straightforward. However, this method also offers positions to be effectively managed across multiple objects and allows for complex position logic.



UML Diagram of Alternative objPos Class Design