COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Jeremy Shi(shi83), Benjamin Semmler(semmlerb)

Team Members Evaluated gande: gandev1, haides35

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 project loop is fairly clear and easy to follow, with each main loop function (initialize(), getInput(), etc.) having its respective code. The initialize function clearly initializes the game board class using dynamic memory allocation, then the player class with a reference to the game board using dynamic memory allocation, and then generates food form the game board. Additionally, they added extra functions (not in recommended workflow), to aid in the coding process, but some variables and functions feel a bit redundant. For example, in the GetInput() function, the line 'gameBoard->getSnakeInput()' uses a newly created function that wasn't reused, and takes even more space/memory than just writing 'if hasChar, getChar'. This is also done again in the drawScreen() function where it initializes variable 'objPos foodPos = gameboard->getFoodPos()' to (I think) make it easier to access the position of food object as it then uses foodPos to access coordinates of the food, but goes back to using gameboard to access symbol of the food. These lines make it a bit more confusing to follow and question how the objects interact with each other, but overall, the code is still clear and understandable.

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.

C++ OOD Pros

- Game functionality is encapsulated in their respective classes, reducing complexity (sometimes) & amount of code (more reusability), increasing readability/understanding, and better maintainability
- Easier for partners/other groups to understand the structure and interaction of the code
- Easier to debug, fix, and add, as classes & functions can be individually tested and developed
- Easier to add more features (like changing from one snake object to the list)

C++OOD Cons

- Can add more complexity when doing simple tasks (like adding food class if not doing bonus can be too much or having to keep track of deleting pointers)
- Can be harder to test how classes interact with each other
- Can be harder to code as much more to consider when classes interact or

- Can be harder to understand if not done right

C Procedural Design Pros

- Code is simple to write and understand.
    - No complex structures
    - No multiple classes that interact with each other
    - Less keeping track of object instantiation and memory management (for pointers)
- Faster to code for more simpler tasks than OOD

C Procedural Design Cons

- A lot of global variables to keep track of
- Harder to scale
- Reduced reusability as it is procedural coding and in PPA3, functions are made with a specific global variable in mind
- Harder to maintain/understand as there is a lot of code and they are all interacting with each other
    - Annoying to debug

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 has concise comments throughout the program, which followed up with clear indentation and line spacing, helps aid in understanding the code more efficiently. The DrawScreen() function has comments on each conditional statement (except for the food) to help understand what it is drawing on the screen. This is similarly done in the reset of the classes, where they include comments for all functions and parts that aren't immediately obvious in what they do.

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 follows proper indentation and adds sensible white spaces/newlines for better readability. Blocks of all conditional statements have their respective newlines and indentation for understanding, and parts like loops, function parameters, operations, all use sensible white spaces to aid in the readability. For example, the nested for loops in drawscreen function have spaces and new lines for each conditional and after each loop.

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)

Only "bug" we could find, is on the exit message after dying to self-collision, it prints:

You Lost the Game

aYou are now leaving the game

This is clearly a typo and has to do with an added a after their newline in MacUILib_printf("You Lost the Game\n\na");

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.

No memory leak found.

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:

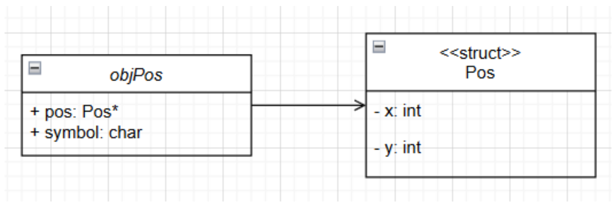
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

No, I think the additional compound object design of objPos class is not sensible. Using Pos struct as a pointer instead of just private x and y integer values in objPos class isn't needed in this project. It complicates memory management, requires more memory and functions to access values, and overcomplicates it. In both our project and of the team we evaluated, in order to access the x and y positions, we used a long line of code 'player->getPlayerPos()->getElement(k).getObjPos().pos->x' where we are now directly accessing the value, instead of returning it from a function ('.pos->x'). Though it works completely fine, and could be useful in terms of having a variable for the coordinate of each object, as well as being able to be used by other classes as well if they also needed a variable for only coordinates, in this project, the extra complexity is not sensible.

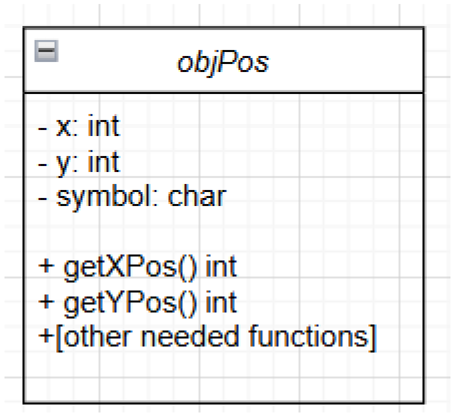
2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. <u>You are expected to facilitate the discussion with UML diagram(s).</u>

Adding the x and y values directly into the objPos class without using a Pos struct to store the coordinates as simple integers is easier to do and manage. It makes it easier because you don't need to create a new Pos Struct and thus no extra pointers and dynamic memory allocation. It could make it more clear when accessing the position variables from outside the class like in the example in the previous part 1. Though we would lose reusability of having a struct object for coordinates that all classes could use, the fact that only objPos uses this struct and the simplicity and efficiency that would come from having x and y values directly part of objPos make the improved design more sensible.

Original:



Improved:


objPos
- x: int
- y: int
- symbol: char
+ getXPos() int
+ getYPos() int
+[other needed functions]