McMaster University

COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members: Aryana Akhlagh Nejat and Dora Mackie

Team Members Evaluated: Shiv Mahida and Adeel Ahmad

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.

It is a bit confusing how the game mechanics and food classes function since the game mechanics class includes things associated with food that could have been done in the food class. They created a food object from the food class inside the constructor of the game mechanics class and probably should have just initialized food in the main project file. Since this creation of the food object is only present in the constructor of game mechanics, they are also unable to use the default constructor as this will never initialize a food object.

A positive feature is how they randomized the '@' spawning using variable *chance*, setting chance to any number between 0-25 and if *chance*==0 then the '@' symbol will print. Also, destructors were made in every class .cpp file, however, in the main project file the destructors were never used, but things were still deleted in the clean up function. It was clear how the myGM and myPlayer pointers were used to access the game mechanics and player class files, however, one was never made/used for the food class.

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:

- Using the C++ OOD approach makes the code easier to modify and add onto. We saw this after creating a food class and were easily able to add onto it and create different types of food that affect the player differently.
- C++ is easier for collaboration as we were able to allocate different class creations to different team members. We were able to work in different files without overriding the other person's code.

- In C++, we have more control over our memory use as we are able to allocate memory onto the heap which we then free once the program is done running, as opposed to using multiple global variables like in PPA3.

Cons:

- The C++ approach is more difficult to debug as there are many files to consider (header files, class files, and the main file).
- Using C procedural uses less memory while it is running, since we are not allocating as much and there are not many files.

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.

There are good comments explaining putting *player* on the heap, as well as on the changed functions and why they made these changes (in game mechanics, how many points to increment score). Their comments highlighted the default constructor, copy constructor, copy assignment constructor and destructor. The player function was well commented and explained the logic behind each *if* statement well.

The draw screen function does not have sufficient commenting as it does not explain what is being printed and why it is being printed. To improve this they should comment at each for loop what they are iterating through to print and explain what the if statements are checking for and why they would continue to print or cease to.

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.

This team made good use of indenting and whitespaces throughout their code. Everything was spaced out very well, which made the code very readable to someone who is not familiar with it. With their print statements, everything printed on a new line so it was readable and not too cluttered. They could have opted to add additional new lines to split up their score and instructions from the game board but it is overall very readable and clear.

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)

This game generally provided a smooth playing experience. However, there were a couple parts that could be improved upon to make the experience even better.

The first thing we noticed was that food was able to generate underneath the snake body. This was most noticeable as the snake got longer and there were more chances for the food and the snake to overlap. The root cause of this error is likely caused by issues in the generateFood function in the food class, and not properly preventing overlap from occurring. To debug this, the team could first confirm this occurs by printing out the coordinate of the food and player positions in a list, maybe slowing down the gameplay to be able to read these by changing the delay value. Then they could potentially use the VS code debugger to run through this portion of the code and check that their open[][] array (functioning as a bitvector) is working properly in order to identify the root cause.

Additionally, there were a couple flaws related to the game exit messages. When you press the spacebar to exit the game, we noticed in the code that a message saying “You exitted” is supposed to appear. However, this does not occur. Additionally, when you win the game, you briefly see a message that says “You win!”, before it is almost immediately changed to say “You lose!” instead. Both of these bugs likely have to do with how the drawscreen function works. For the first error, the message is being printed and then immediately cleared because the exit flag is set to true. The team should think of alternative ways to implement their *if* statements in the drawscreen to avoid this. Additionally, the second issue likely stems from the fact that while the drawscreen function stops when you win, the snake still technically “moves” and self-collides. Again, the team should reconsider the drawscreen, and overall gameplay logic to prevent these errors and make the game run more smoothly.

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.

We found no memory leaks when checking this code with Malloc Stack Logging on macOS.

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:

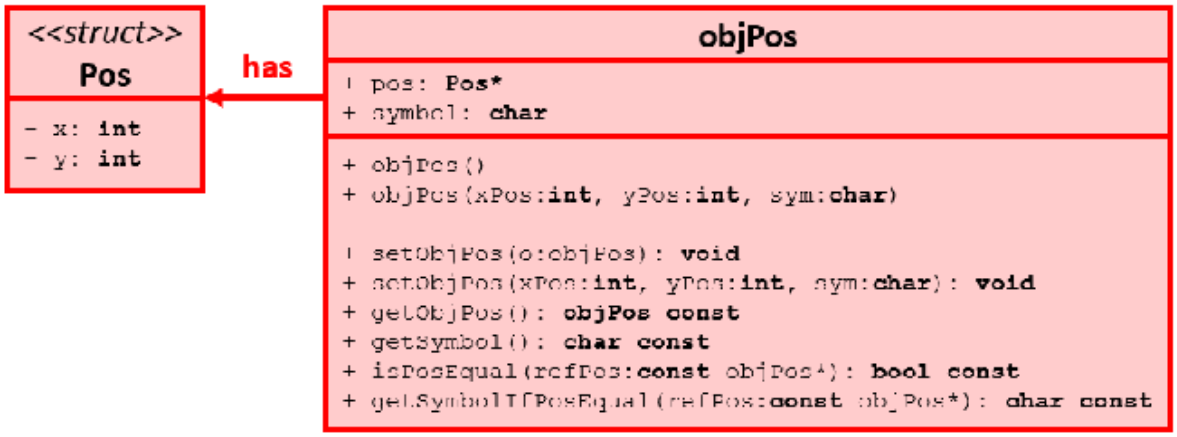
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

No, the design of the objPos class is not sensible, as the only time we made objects in the objPos class was when we needed to focus on a specific element of an object in the objPosArrayList class. Since the position including the x and y coordinate values is a struct, there were multiple extra steps we needed to take to access these values. We needed to first reference the objPos class or create a new object as part of the objPos class, then access the position struct whenever we wanted to access the x and y coordinates of an element, which was a bit convoluted.

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>

An alternative method would be to include the coordinates within the actual objPos class itself, instead of including them inside of a struct. This would make them much easier to access and remove the unnecessary step of having to take the value from the struct itself.

Original UML Diagram



New UML Diagram with changes highlighted

objPos
- x: **int** - y: **int**
+ objPos() + objPos(xPos: **int**, yPos: **int**, sym: **char**) + setObjPos(o:**objPos**) : **void** + getObjPos(xPos: **int,** yPos: **int**, sym: **char**): **void** + getObjPos(): **objPos const** + getSymbol(): **char const** + isPosEqual(refPos: **const objPos***): **bool const** + getSymbolIfPosEqual:(refPos: **const objPos***): **char const** + getXPos(): **int const** + getYPos(): **int const**