COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members zengw16, sharml12

Team Members Evaluated yanga82, dugha

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.

   Unfortunately, lot of the main logic is commented out, but we can see how most of the main functions would work. For example, it's clear how their drawscreen() function would have worked, with iterating through cells and checking for an edge, player part, or food and printing out the correct thing. It appears to be collecting items in a string similarly to ppa3, which is nice. All of the key ideas are visible.

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:

//The encapsulation allows for better code separation & easier debugging + scalability.

//Since we don't need to manually allocate/free memory, the risk of memory leakage is much lower.

C++ OOD Cons:

//Adding in the classes, constructors, destructors etc and keeping track of inheritance means more planning before writing any code.

//In general, it's harder to grasp ood concepts, and may cause a lot of bugs initially.

C Procedural Design Pros:

//Implementing code via just functions & global variables is a lot easier; there's not really much abstraction here.

//With the lack of classes, it's much faster to write out a prototype; for example, adding a wrap around condition in PPA3 is faster without having to implement a class for it and watch inheritance.

C Procedural Design Cons:

//Without encapsulation, you'd be repeating a lot of logic for managing items, the board, etc.

//Manually allocating and freeing memory increases the risk of memory leaks or segfaults.

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.
    a. Overall the code offers insufficient comments. They had a few comments about the constructors and destructors, and few regarding input processing and FSM logic. There are confusing commented out sections that seems to be a completed section of code? (ie. Project.cpp lines 81-89). Many important functions such as movePlayer, lacked detailed comments explaining their 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.
    a. The code follows consistent and good indentation and functions are separated for better readability. There is an inconsistent use of spacing around functions such as in objPosArrayList.cpp line 45; aList[0]=thisPos; could be more readable as aList[0] = thisPos;. Additionally, the updated aList[0] = thisPos; would be using consistent spacing formatting.

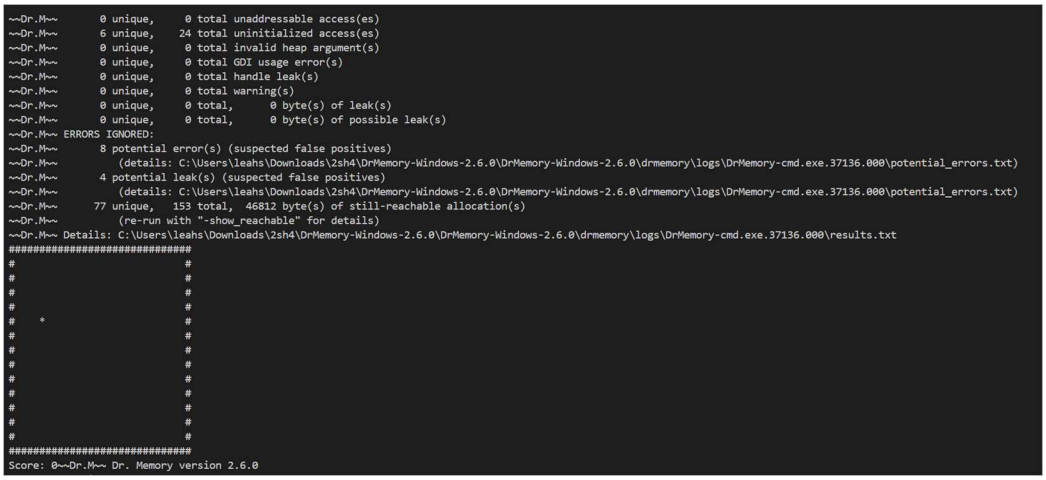
Peer Code Review: Quick Functional Evaluation

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)

    a. Our peers provided an incomplete snake game. The player position doesn't update to any of the wasd or WASD inputs and does not have an exit button that functions. The updatePlayerDir does not handle inputs. There is no food generation code, or score keeping logic. As the code doesn't compile into a functioning Snake Game due to lacking many key aspects of the requirements, there is not a lot that we could recommend for the team to deploy.

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, the game does not cause any memory leak. The report is below:



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:

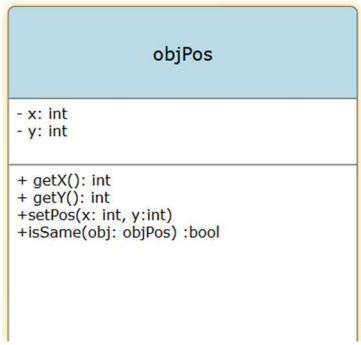
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

   The compound design is somewhat sensible, given the context. On one hand, it's easy to extend the Pos struct if any more movement features are needed. The fact that the Pos struct is nested in there also sort of 'separates' the actual position data x , y from the rest of the objPos class, so again it's easy to make changes/use Pos in things that require position data like distances or maybe even collisions.
   However, on the other hands in this case, since Pos isn't really used for much else other than holding position data, we may not need a whole extra struct for it. The 'extra' struct also results in an extra step when calling location, instead of being able to just directly say something like objPos.x, which can cause syntax errors if not implemented carefully.

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).

Having Pos separated into a different class entirely would be (in our opinion), quite annoying to integrate while alternating between class use, and would likely cause syntax errors + be unintuitive to someone who aren't the developers to use.

Thus, the compound design works. However, as described earlier, since Pos only really holds location data x and y, it could be further compounded within objPos:



This way, x and y are members of the class directly and not in a nested struct.

The issue with this implementation is that extending features using location data would probably result in messing with the objPos class directly.