COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Jumana Ismail, Viha Tripathi

Team Members Evaluated pated201, wur99

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 program loop is organized into distinct phases (GetInput, RunLogic, DrawScreen, and LoopDelay) which makes it easy to understand the overall flow. The relationships between each object are also pretty clear; for example, player->movePlayer() uses direction data from gameMechs->getInput() to update the snake's location, while food->cook() remakes food locations based on collision checks with the snake's body. This is something you can interpret without examining the code itself too much. Additionally, the use of descriptive method names such as getSnake() and setGrid() helps us understand the purpose of each method. That said, some aspects of the logic could be rearranged for improved readability and to avoid confusion. For example, I noticed that the handling of game-ending conditions is spread across different parts of the loop. For example, you checked LoseFlag and ExitFlag in RunLogic and then checked it again in CleanUp. Although this does not affect the code's functionality, this duplication can lead to inconsistency in the code if changes are made in one part but not the other.

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 the C++ OOD approach:

- Better readability and the good is better structured due to encapsulation where we have separate classes for each data such as Food, Player, etc.
- Code can be reused in different files and could add new functions to it easily

Cons of the C++ OOD approach:

- Complex, where we have an entire class just for a small project
- Higher memory

Pros of the C approach in PPA3:

- Simple where we just need to use global variables and functions rather than classes for each data
- Faster to implement due to the code being all in one place

Cons of the C approach in PPA3:

- Code is harder to manage when its a bigger project as there will be tons of global variables and functions all in the same files
- More likely to have bugs due to many global variables
- Harder to debug due to the many global variables and functions in the same file

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 demonstrates a good effort in providing descriptive comments for most of the functions. For example, the CleanUp() function includes a clear comment on its purpose to handle cleanup operations, and the RunLogic() function has comments throughout the whole process (moving the player based on direction, checking if the player has eaten food and if index is not -1, then making some food). These comments are straightforward, making them accessible to both experienced programmers and readers with limited technical expertise.

Apart from that, one change could be incorporating more comments in between blocks of code to more specifically explain what each code action is doing and how it was done so that viewers can see your thinking process. For example, adding details explaining the purpose and logic of how player movement is calculated or why specific methods are used in a specific sequence, can provide a more clear view of your thought process. Along with this, there is some unnecessary code that could potentially lead to confusion. I noticed that you included a win condition which seems out of place for a game like Snake, where the primary goal is survival rather than winning. Once again, this does not affect the functionality of your code, but may confuse someone who is looking through the code so it would be best to get rid of it.

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 has good readability where it follows good indentation, sensible white spaces, and deploys new line formatting. As in across all the files there is a constant format of white spaces added in lines of code that make sense such as when declaring global variables and when using math operations. Additionally, there is a constant use of good formatting of indentation such seen in if statements, for loops and in constructors and destructors. Lastly, newline formatting is used throughout the coding files to make the code visually appealing and easy to read such as adding a \n after the MacUprintf statements.

Peer Code Review: Quick Functional Evaluation

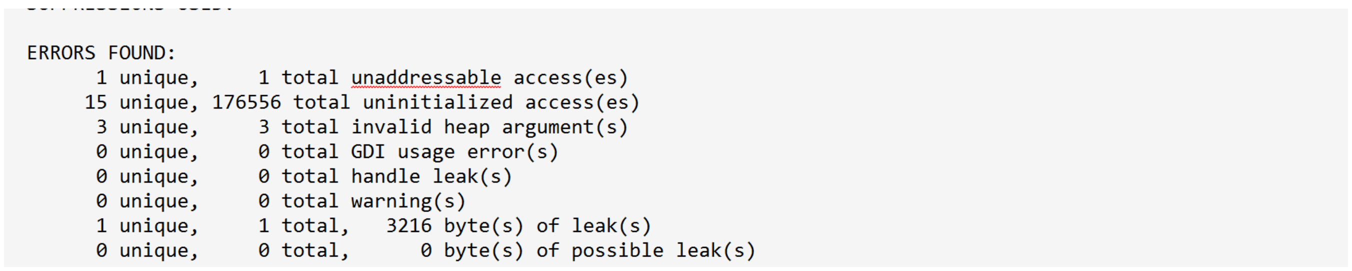
1. **[3 marks]** Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just technical user feedback)

The Snake Game offers a generally smooth and playable experience, with no major issues that disrupt the gameplay flow during normal conditions. A notable feature is the incorporation of adjustable game speeds, which allows players to tailor the game to their comfort level. This demonstrates thoughtful consideration for user experience. Along with that, there are instructions on the game screen which provides a good playing experience.

Some minor things that could be changed is the game board. There is a piece missing in the corner of the gameboard which could be fixed for the sake of aesthetics and user experience. Another suggestion is displaying dynamic game data such as the snake's coordinates and symbol so the player knows where they are at all times as well as what symbol their player is so they are able to differentiate between the snake and the food.

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 snake game does cause memory leaks. There is 1 total, 3216 byte(s) of leak(s). The possible root causes of the memory leakage according to the report is that the program is trying to read memory that has not been initialized and the locations of this issue is in the following functions: GameMechs:: setGrid, GetInput, and main. The second possible cause may be from unaddressable access and these can be from the destructors used due to memory being freed more than once. Lastly, another possible root cause is invalid heap argument to free meaning that the program may be trying to free memory that's already freed or not dynamically allocated.



```
ERRORS FOUND:
      1 unique,     1 total unaddressable access(es)
     15 unique, 176556 total uninitialized access(es)
      3 unique,     3 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      1 unique,     1 total,   3216 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
```

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?

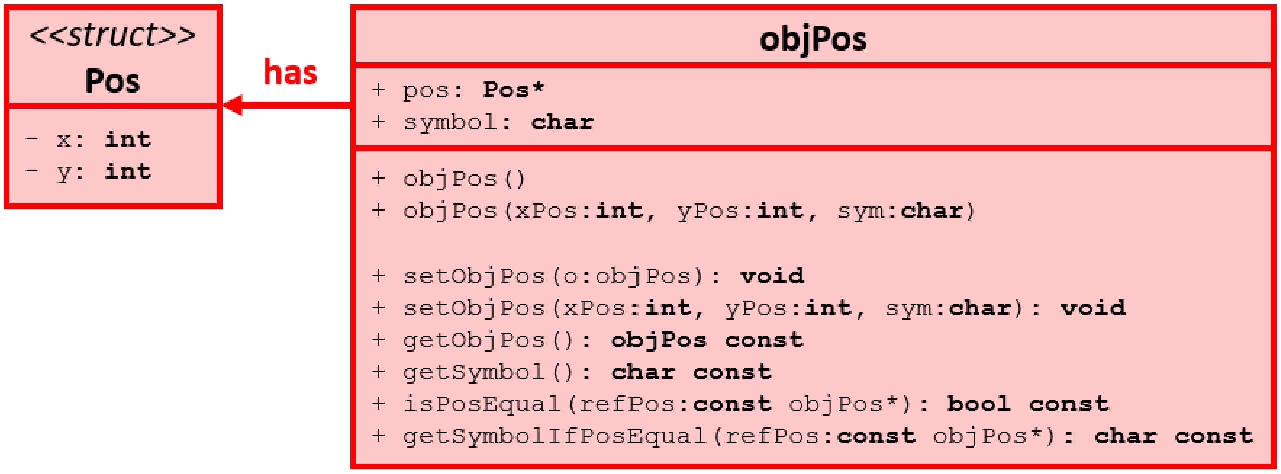
I think the objPos class design is practical for this project. The separation of positional data into a struct makes the code clean and easier to maintain. For instance, all position-related data is handled by the Pos struct, while the objPos class would contain additional attributes like the symbol. That keeps the code organized, ensuring that position logic changes do not spill over to other parts of the class.

This also makes the code easier to work with. For example, when checking for collisions or updating positions, the pos->x and pos->y make it clear what data is being manipulated.

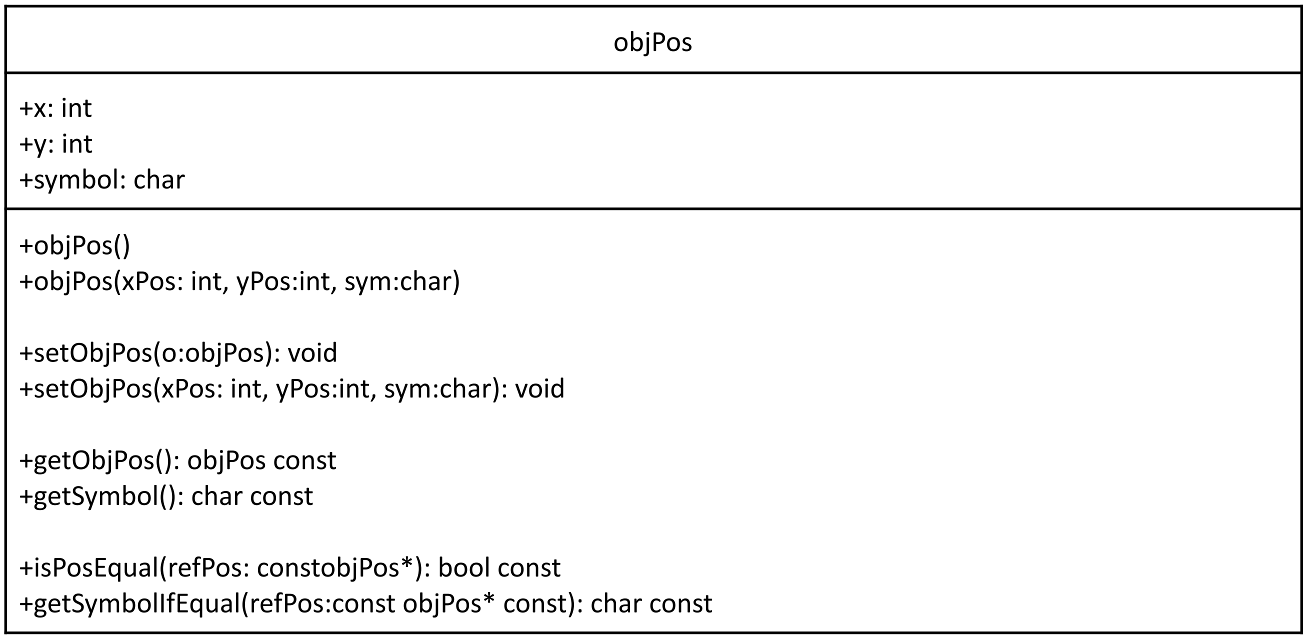
2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. You are expected to facilitate the discussion with UML diagram(s).

An alternative objPos class design that I believe is relatively counterintuitive that the one in this project is by having the variables x and y as integer data members of objPos class instead of having them as members of the struct Pos. The changes of the UML diagram is the following:

Old UML:


<<struct>>
Pos
- x: int
- y: int
has
objPos
+ pos: Pos*
+ symbol: char
+ objPos()
+ objPos(xPos:int, yPos:int, sym:char)
+ setObjPos(o:objPos): void
+ setObjPos(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

Updated:



objPos

+x: int
+y: int
+symbol: char

+objPos()
+objPos(xPos: int, yPos:int, sym:char)

+setObjPos(o:objPos): void
+setObjPos(xPos: int, yPos:int, sym:char): void

+getObjPos(): objPos const
+getSymbol(): char const

+isPosEqual(refPos: constobjPos*): bool const
+getSymbolIfEqual(refPos:const objPos* const): char const