COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members girgik1/tsangb8

Team Members Evaluated esmailm/

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

*This peer evaluation is done on the version in folder **course-project-heap-heap-hooray\Final Ver\Final\course-project-heap-heap-hooray-01a826e5f433c5a613913bb759223bf74b3bc9ae***

The repository contained a minimum of 6 versions of the code, some of which are within another version of a code in a separate subfolder. Some of these codes do not compile as they are missing entire .vscode subfolder and the launch.json files. Some of these do get made but do not run properly (run and nothing happens and terminal prompts a new command). This version was one of the ones named with final and runs best. We are doing this peer review to the best of our ability, if there are any concerns and confusions, please contact us anytime.

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 in project.cpp. They have declared a global objPos object which was directly conflicting with the project outline (declared on line 17). The objected it self is initialized inside the main loop's void Initialize(void) with (0,0,'#'). They then only ever use this object again in the main loop's DrawScreen(void) function where they only referenced its symbol ('#'). This mistake was completely avoidable by using a local variable inside the DrawScreen(void) function. Other than that, the code follows the outline provided, besides one redundant line (identical lines) on line 91-92 in the void getInput(void) function of the main loop.

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
  - More customizable
    - o If you wanted to change the variables (board size, starting conditions/positions, etc.), implement new features and functions (example of developing the bonus features), or make any changes in general, C++ OOD approach is a lot easier to implement. The program runs by referencing variables and functions stored within classes/objects and changing said

class/object would efficiently change very time that class/object is called) In a C procedural design approach in PPA3, we would need to hunt down where the scattered functions and variables and change them

  - More readable
    - Related variables and functions are neatly packed into respective classes in the C++ OOD approach.
  - Reusability
    - The objPos and objPosArrayList are reused numerous times in our version of the game but would proof to be more difficult in C procedural design approach as they lack the organization C++ OOD offers.
- Cons
  - Possible overengineering/packaging
    - This can be seen by the group's boardObj object being created simply to hold a single '#' character

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.

While there were comments throughout the code, most of the comments implemented were to comment out blocks of unused code or test code. Making the overall objects and files very bloated with huge walls of commented code in the middle of the actual code. This commenting persists throughout all cpp files in this version. Code can be improved by deleting said commented out codes. Other than that the code function seem to follow the project guidelines.

There is however a huge issue in terms of their folder management as it is near impossible to figure out which version is the most updated. I have put a statement at the top of the documents before the evaluation part begins stating all the issues above.

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.

While there were comments throughout the code, most of the comments implemented were to comment out blocks of unused code or test code. Making the overall objects and files very bloated with huge walls of commented code in the middle of the actual code. I would delete those segments of code and add more comments explaining the less intuitive parts of the objects such as what the getters and setters are doing and how the mechanics function exactly.

Aside from that, indentation and white space use made program legible.

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)

Snake game was fully functional, would've been nice to see more interface such as how to terminate program, overall, no major bugs found.

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.

Proper memory management implemented; no memory leaks found on DrMemory.

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 it was unnecessary to include the Pos struct in the objPos object; this is because the Pos struct could have been implemented as two separate object variables, x and y. While it does not affect much in our code, it does make accessing these variables more tedious as we must reference the struct and reference the variable in the struct.

While it is not useful for our implementation in objPos, it has uses in other fields such. For example, if there are many variables to keep track of and you don't require any class specific features (methods, polymorphism, etc.) struct would reduce the amount of object variables in our code and organize them into separate structs. There is also the fact that structs are public which can be an issue but in our case the Pos struct was public and meant to be accessible anyways, so we could've just had two class variables instead of a struct.

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>

objPos
+ int x + int y + char symbol
+ objPos() + objPos(int xPos, int yPos, char sym) +~objPos() +objPos(const objPos &p) +objPos& operator=(const objPos &m) +void setObjPos(objPos o) +void setObjPos(int xPos, int yPos, char sym) +objPos getObjPos() const +char getSymbol() const +char getSymbolIfPosEqual(const objPos* refPos) const +bool isPosEqual(const objPos* refPos) const

Shown above is the UML diagram of the proposed improvement to the objPos class. As we can see, there are no major changes aside from the fact that we have broken down the struct that was previously found in the class into two int variables x and y.

Within the code itself we would also have to change the getter and setter implementations as well to compensate for the lack of the struct, additionally we would want to instantiate these on the heap since we want to use these functions

This reduces memory usage since struct takes more memory, streamlines the coding process and improves code readability when calling the position x and y since it allows the programmer to only call on objPos.x and objPos.y instead of objPos.Pos->x.