COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Shaista Mohammed - mohas40, Reem Basiouny - basiounr (reesh)

Team Members Evaluated wu744 & huiy13 (hedden3rd-floorsouth)

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 runs with typical object-oriented programming features. The references to different classes and objects are easy to interpret through descriptive variable and pointer names. The correct items are placed in the correct functions based on their purpose (initialize, draw screen, etc.). The additional method of "getElementMap" aids in cleaning up the code in "Draw screen". Overall, the main function is clean and efficient.

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 C++ OOD:

- Allows a more modular design and organizes code that has related functionality into classes, making the program easier to understand and organized with a clear purpose to each class.
- The main logic function of the program stays concise, allowing detailed operations to be handled by specific classes.
- Allows the management of multiple similar items (like food in a game) without repeating the same code for each one. For example, you can generate multiple food with different attributes using one class template.
- Allows sharing data safely between parts of your program by passing specific references or pointers. The classes allow control of what is kept private or public, so only the intended parts of the program can modify important information.

Cons of C++ OOD:

- Due to manual memory management in C++, it is easy to have issues with regards to memory leaks, dangling pointers, etc.
- Debugging and making changes to code can be more difficult as different parts of the code (classes and objects) are interdependent.

- Requires careful planning and a very thorough understanding of the classes created by all programmers (if the program is collaborative software) to manage interactions between multiple classes, which can become complex as the project grows.

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 offers a sufficient number of comments, each explaining functionality of the corresponding code. Spacing between comments and code could have been more consistent to aid in code readability and avoid clustered code-blocks. For example, at the end of Run-logic and the beginning of Draw-screen. Overall, the functionality and OOD methods in the code are understandable through the comments.

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 follows good indentation practices, as the nested blocks are consistently indented, making the structure of the code easy to follow. Appropriate whitespaces are used between functions and different sections of the program are separated logically. Also, newlines are used effectively to improve readability, such as in the DrawScreen function, where nested loops are clearly structured, and outputs are separated by newlines.

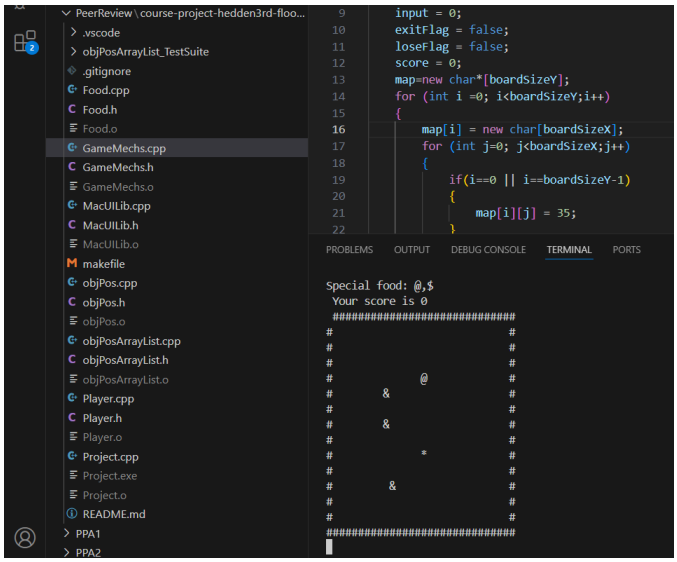
However, one improvement suggested would be breaking up long lines of code into multiple lines. For example, a line like “mainGameMechsRef->setElementMap(playerPosList->getTailElement().pos->y, playerPosList->getTailElement().pos->x, 32);” in the movePlayer class forces us to scroll horizontally to view the entire line, which can be avoided. Also, a minor change would be to add some white spaces between longer blocks of code when initializing local variables before the conditionals, such as in movePlayer. This would avoid code looking like a “paragraph”, all squished together. These minor adjustments would ensure the code is formatted for even better readability.

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 provides a smooth playing experience overall. However, the game screen does not provide any player instructions to tell the player what control buttons to press for each direction or how to win/self-end the game. A bug that was discovered was the upper left corner of the border prints a space instead the intended hashtag to complete the boarder. A possible root cause could be found in the GameMechs class that is referenced to in the Draw-screen function when the code attempts to prints the boarder. There could be issues

with the indexing or print conditions that instructs the program to place a “space” at the coordinates [0,0]. Other than this bug and the missing player instructions, the game is smooth and essentially bug-free.



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 Snake Game does not cause any memory leakages. The memory leakage report generated resulted in a conclusion of “0 leaks for 0 total leaked bytes”, which indicates effective memory management.

```
(base) shaistamohammed@Shaistas-MacBook-Pro course-project-hedden3rd-floorsouth % export MallocStackLogging=1
(base) shaistamohammed@Shaistas-MacBook-Pro course-project-hedden3rd-floorsouth % leaks --atExit --list -- ./Project
leaks(14524) MallocStackLogging: could not tag MSL-related memory as no_footprint, so those pages will be included in process footprint - (null)
leaks(14524) MallocStackLogging: stack logs being written to /private/tmp/stack-logs.14524.1054bc000.leaks.pM8EBG.index
leaks(14524) MallocStackLogging: recording malloc and VM allocation stacks to disk using standard recorder
Project(14525) MallocStackLogging: could not tag MSL-related memory as no_footprint, so those pages will be included in process footprint - (null)
Project(14525) MallocStackLogging: stack logs being written to /private/tmp/stack-logs.14525.104574000.Project.
Project(14525) MallocStackLogging: recording malloc and VM allocation stacks to disk using standard recorder
leaks(14528) MallocStackLogging: could not tag MSL-related memory as no_footprint, so those pages will be included in process footprint - (null)
leaks(14528) MallocStackLogging: stack logs being written to /private/tmp/stack-logs.14528.102604000.leaks.3q13LT.index
leaks(14528) MallocStackLogging: recording malloc and VM allocation stacks to disk using standard recorder
Process 14525 is not debuggable. Due to security restrictions, leaks can only show or save contents of readonly memory of restricted processes.

Process:         Project [14525]
Path:            /Users/USER/Documents/*/Project
Load Address:    0x1040fc000
Identifier:      Project
Version:         0
Code Type:       ARM64
Platform:        macOS
Parent Process:  leaks [14524]

Date/Time:       2024-12-05 04:34:07.465 -0500
Launch Time:     2024-12-05 04:34:03.510 -0500
OS Version:      macOS 14.6.1 (23G93)
Report Version:  7
Analysis Tool:   /Applications/Xcode.app/Contents/Developer/usr/bin/leaks
Analysis Tool Version:  Xcode 15.4 (15F31d)

Physical footprint:         4097K
Physical footprint (peak):  4097K
Idle exit:                  untracked
-----

leaks Report Version: 3.0
Process 14525: 297 nodes malloced for 352 KB
Process 14525: 0 leaks for 0 total leaked bytes.

leaks(14528) MallocStackLogging: stack logs deleted from /private/tmp/stack-logs.14528.102604000.leaks.3q13LT.index
leaks(14524) MallocStackLogging: stack logs deleted from /private/tmp/stack-logs.14524.1054bc000.leaks.pM8EBG.index
(base) shaistamohammed@Shaistas-MacBook-Pr                    ct-hedden3rd-floorsouth %
```

Screenshot

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 believe that the additional Pos struct is not the most sensible design. It acts as an extra step to access the x, y, and symbol data when these variables could be made directly memebers of the already made objPos class. It would be more efficient and just as functional. Additionally, typedef is not a part of the C++ based design. Overall, it would be cleaner and more efficient without the compound object design.

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).

To improve the object design, I would remove the Pos struct and add x and y coordinates of type int as private member variables within the objPos class, which simplifies the class design by getting rid of dynamic memory allocation and pointer dereferencing. Now, instead of calling refPos->pos->x, for example, you could simply call refPos.x and refPos.y , which is simpler. The symbol can be used as a private member so that it can't be modified directly from outside the class. Three getter and setter methods each would be created for the three attributes (x, y, and symbol) and would be created publicly so other classes would be able to use them in the program. This

design is a simpler, easy to use, and more efficient solution for the objPos class and maintains object-oriented practices.

objPos
- x: **int** - y: **int** - symbol: **char**
+ objPos() : void + objPos(x: int, y: int, sym: char): void + getX(): int const + getY(): int const + getSymbol(): char const + setX(xVal: int): void + setY(yVal: int): void + setSymbol(newSym: char): void + isPosEqual(refPos: const objPos*) : bool const + getSymbolIfPosEqual(refPos: const objPos*): char const