COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members kim600 wang562

Team Members Evaluated golwalaz singp32 (Team name = grenglins)

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.

   Yes, we can easily interpret how the objects interact with each other. They make full use of all the functions provided to them through the Player and GameMechs class throughout the main program loop, such as calling the movePlayer and updatePlayerDir in the main logic and using object methods such as getSize to get the player's size when printing. The program's structure follows the provided UML to establish clear links between each 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:
   - Encapsulation: Keeps data and methods organized and secure within objects, reducing risks of unintended interactions. Private and public components of each class are compartmentalized to eliminate the risk of accidental usage or interactions.
   - Modularity: Easier to manage and extend the game components (e.g., Player, Items, Board) due to clear boundaries. Programmers can add, remove, or modify existing elements and functions to improve on the object's overall abilities.
   - Reusability: Code such as collision detection and random generation can be encapsulated in classes, promoting reuse across projects

   Cons:

   - Complexity: Object-oriented design can be harder to implement and debug. With additional features like overloading and polymorphisms, an inexperienced programmer may get confused or make costly mistakes.
   - Overhead: Since C++ addresses more syntax issues and has additional QoL features in the compilation process compared to C, there may naturally be more overhead compared to a more lightweight C program.
   - Setup Effort: Requires more upfront design to define classes, relationships, and interfaces, which can be challenging in time-constrained scenarios

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 sufficient comments that efficiently describe lines of code that may need more clarity. The comments also describe the purpose of all the functions and very clearly demonstrates the connection and workflow between the classes and main logic.

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.

   Yes, the code has proper formatting and contains white spaces/newlines where it makes sense, enabling good readability. For example the getter and setter methods of classes are separated by newlines to illustrate separation

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)

   Yes, the game offers a smooth, bug-free playing experience. The game works perfectly as intended, moving the snake and implementing the wraparound while increasing the length when a food is eaten. The special food was also implemented correctly, decreasing the score by 4 everytime the special food is eaten.

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 memory leak, as seen below in our DrMemory report of the project.

```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       7 unique,    16 total uninitialized access(es)
~~Dr.M~~       1 unique,    64 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total GDI usage error(s)
~~Dr.M~~       0 unique,     0 total handle leak(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       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?

   No, we believe the compound object design of the objPos class is not entirely sensible. It adds unnecessary complexity to access position attributes like x and y. Instead of directly accessing these values through the object, the current design forces multiple calls (e.g., getElement(i).pos->x), which is tedious and less intuitive. The current implementation also allows for x and y to be modified directly after calling it, which does not follow the encapsulation object-oriented design principle.

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).

   We would improve the design by making x and y be private members in the objPos class with getters and setters. This allows the position to be called more intuitively using getter methods, and any modifications to the position can be done using setter methods.

objPos
- xPos: int - yPos: int - objSymbol: char
+ objPos() + objPos(xPos:int, yPos:int, sym:char) + ~objPos() + objPos(other: &objPos) + operator=(other: &objPos) + setObjPos(o: objPos): void + setObjPos(xPos: int, yPos:int, sym:char): void + getObjPos( ): objPos const + getSymbol(): char const + getX(): int const + getY(): int const + setSymbol(sym: char): void + getX(xPos: int): void + getY(yPos: int): void

+ isPosEqual(refPos:const objPos*): bool const
+ getSymbolIfPosEqual(refPos: const objPos*): char const