COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members Mohit Chopra, Aaditya Anil

Team Members Evaluated wangd148, rizvia21

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 **30 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.

- Their code only does srand(time(NULL)) at initialization, while not incorrect, reduces the randomness of the program due to only one seed being generated, compared to calling srand at every generation which constantly changes the seed.
- Nearly all program logic occurs in the movePlayer() function, which while it works, it is difficult to interpret where food generation or collision is occurring in the program and reduces modularity. Functions such as lose condition, food generation and collision should be called as separate functions in the run logic block of the main code to enhance readability.
- Some methods in certain classes (eg. Player class) are declared but never used (eg. getPlayerPos), which, while not necessarily wrong, can be confusing at times considering said methods are declared but never called or used.
- DrawScreen, GetInput and CleanUp blocks are done well, easy to understand logic behind them.
- CleanUp deletes heap objects to ensure no memory leaks.

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:
   - increased modularity, makes it easier to change and add features to code without messing up other components
   - Allows for scalability and reusability, reduces need to create many functions with similar code.
   - Readability, it is not necessary for the reader to always know the logic being each function, using OOD only exposes necessary data and components to the user.

   Cons:
- Uses of more inheritance in multiple classes in higher complexity tasks can be considerably more complicated to implement, as certain classes will rely on numerous additional subclasses, some of which may be excessive/unnecessary.

- If object or class and inheriting classes are incorrectly implemented, debugging will result in “explosion” of the object in question and its class, the inheriting class, and all interactions between the two classes that take place throughout the code’s execution, cluttering debugger with garbage.
- Not always needed for certain tasks, especially if the task is simplistic in nature.

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.

- Program has many comments throughout explaining the basics of what each function does and comments throughout explaining how the functions work.

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.

   Most of the program follows good formatting for readability, however there are some cases where there needs to be better indentation.

```
    else
    {
        objPos newPos(newX, newY, '@');

    // Check for collision with a total of 5 foods
    for (int i = 0; i < 5; i++){
    if (newX == mainGameMechsRef->getFoodpos(i).pos->x && newY == mainGameMechsRef->getFoodpos(i).pos->y) {
        // Eat the food: Insert new head, don't remove tail

        // if eat normal food
        if (mainGameMechsRef->getFoodpos(i).getSymbol() == '*'){
            playerPosList->insertHead(newPos);
            mainGameMechsRef->incrementScore();
        }
        else{ // if eat special food
            playerPosList->insertHead(newPos);
            playerPosList->removeTail();
            mainGameMechsRef->incrementScore5();
        }
        // Generate new food
        mainGameMechsRef->generateFood(playerPosList);
        flag = 1;
    }
    }
```

   In this block from the player.cpp files movePlayer function, the for loop and all its contents should be indented and the if statement within the for loop should be indented. The generateFood function from the foodbin.cpp file also faces the same problem, where loops and conditionals need to be properly indented.

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)

   While running the code, we observe no buggy features. Ways to debug would be running the debugger and making sure the values are as expected and using print statements to see certain values as the code runs.

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 memory leaks found

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      9 unique,   124 total uninitialized access(es)
      0 unique,     0 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)
      0 unique,     0 total,      0 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 yours, reflect on the following questions:

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

Though the compound design of the objPos class does work as intended with little to no bugs or issues, it is not the most sensible nor ideal means of identifying an object's x and y coordinates. The Pos struct is simplistic in function in that it simply stores the x and y coordinates of the objPos type object. This works, though it introduces unnecessary complexity to the objPos class, which should be avoided when dealing with OOD. Instead, a more sensible design would be to simply creating an x and y member in the object class itself, which would make the class less complex (does not need a struct), as well as more modular (given such a scenario occurs). The additional Pos structure is less sensible since it introduces additional complexity to the code, rather than simply implementing Pos's function into the class itself.

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.

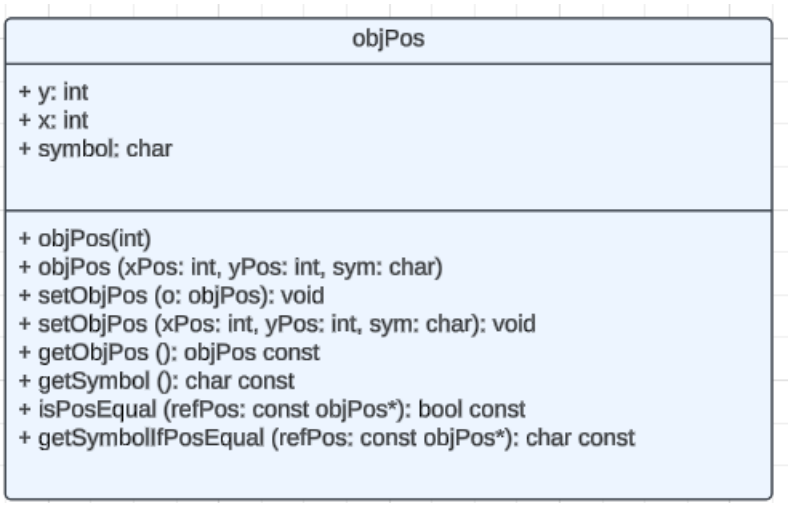
An alternative class design for objPos would be to integrate an x and y member directly into the class upon the class's construction, as mentioned above. By doing this, you would not only remove the need for a struct in objPos and the associated added unnecessary complexity, but also make the overall code and use of x and y coordinates more modular should you need to change anything, since as opposed to changing the struct in the class, you would simply need to change the class itself. An example of this improvement can be seen in the figure below,

```
class objPos {
private:
    int x;          // X-coordinate
    int y;          // Y-coordinate
    char symbol;    // Object's symbol

public:
    objPos();  // Default Constructor
    Codeium: Refactor | Explain | ×
    objPos(int xPos, int yPos, char sym);  // Parameterized Constructor
     Codeium: Refactor | Explain | ×
     ~objPos();
    objPos(const objPos& other);
    objPos& operator=(const objPos& other);

    void setObjPos(int xPos, int yPos, char sym);  // Set position and symbol
    char getSymbol() const;  // Get symbol
    bool isPosEqual(const objPos& refPos) const;  // Check if positions are equal
    char getSymbolIfPosEqual(const objPos& refPos) const;  // Get symbol if positions are equal
};
```

as well as the associated UML diagram with the implemented solution.



The following is a UML diagram of the improved objPos class