COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members yu321-SoDone lin388-mcmaster

(Super awesome team name)

Team Members Evaluated pateh148

(0 memory leaks)

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.

   Examining the main logic in the program loop, the intended outcome of each logical operation (function call, conditional statement) and object iteration was clear and straightforward. Many of the underlying logic is done within object methods, making the main program loop easy to understand. A unique programming design choice is noticed *within the conditional statements* where the program calls getter methods to obtain values (in for loops, if blocks, etc.). A positive effect of this is that the program is very memory-lean as the values obtained from the getter methods are deleted from the stack as soon as the condition is evaluated. This design choice also supports OOD principles by respecting encapsulation through the use of getter methods. A negative effect from this approach is that more computational resources are used as the getter methods need to be called for every loop iteration/conditional statement, where these values could have been stored in a variable for reuse. This can be noteworthy when getters are combined in a chain, like in the example below:

```
for (int k = 0; k < player->getPlayerPos()->getSize(); k++)
{
    if (i == player->getPlayerPos()->getElement(k).pos->x && j == player->getPlayerPos()->getElement(k).pos->y)
    {
        board[i][j] = player->getPlayerPos()->getElement(k).symbol;
    }
}
```

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 Approach in the Project (Relative to C Procedural in PPA3):
   - Encapsulation: Classes like Player and GameMechs safeguard data and behaviour, reducing unintended access.
   - Modularity and Reusability: Objects like objPos and objPosArrayList are reusable and better organized

compared to structs and functions in a procedural approach.
- Scalability: New features, like multiple food items or special behaviours, are easier to implement compared to PPA3's design.

Cons of C++ OOD Approach in the Project (Relative to C Procedural in PPA3):
- Complexity/Debugging: Concepts like references and dynamic memory add a learning curve, making debugging harder than PPA3's simpler flow.
- Memory Management/Overhead: Constructors, destructors, and object lifecycles introduce extra complexity and potential pitfalls absent in PPA3's direct approach.

Commented [JL1]: might be too long

In general, the code is much more readable, especially in the main Project.cpp file, compared to if the project used a procedural design approach. The drawback to this is there are many "nested references" to other objects/classes (for example, player->getPlayerPos()->getElement(k).pos->x) which can complicate things if someone wants to see/modify the underlying logic/code or if debugging is required.

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 provides a reasonable level of comments, explaining key methods, constructors, and logic blocks. However, while some methods are well documented, others, such as insertHead() or removeHead() in objPosArrayList.cpp, lack detail on explanations of how the list objects shift to add/remove the head object, reducing consistency. Despite array lists being covered in the course material, more granular comments should be added to ensure emphasis on clarity and maintainability. Despite this shortcoming, the rest of the code is reasonably commented and allows for others to easily grasp the logic behind the code functionality.

Commented [JL2]: inputs are self-explanatory, no need for 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, uses sensible white spaces, and deploys newline formatting effectively for readability. Upon review, no dominant shortcomings in these aspects were observed that would require immediate attention. The current formatting aligns well with the requirements for clarity and maintainability.

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 mostly provides a smooth playing experience; however, a few issues detract from its

functionality. The special food feature appears to have no discernible effect beyond character oscillation, suggesting a missing or ineffective implementation of its intended bonus behaviour. Additionally, the exit game message is identical to the lose game message when the game is stopped by force quit or losing, and neither custom message remains displayed on the screen after the game ends (i.e. the custom message is overwritten by clearScreen() preventing the player from seeing it). Furthermore, the objPosArrayList class does not implement exception handling for out-of-bounds accesses, as noted in the test cases, which could lead to undefined behaviour or crashes during gameplay if indices exceed array bounds. To address these, clarifying the special food logic in the Player::movePlayer function, customizing exit and lose messages in DrawScreen, retaining them on screen post-game, and implementing exception handling in objPosArrayList::getElement and similar methods are recommended debugging steps.

Commented [JL3]: i.e. the custom message is overwritten by clearScreen() preventing the player from seeing it

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 not appear to cause any memory leaks. All dynamically allocated objects, such as GameMechs, Food, and Player, are properly deallocated in the CleanUp function. Additionally, the destructors in classes like objPosArrayList, objPos, and Food ensure that heap-allocated memory is released. This adherence to memory management practices aligns well with the requirements for avoiding leaks and maintaining efficient resource use.

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?

   The compound design of the objPos class with the additional Pos struct is, in our opinion, not sensible. Since all the data members in objPos are already in the public scope and none of the objPos methods functionally require x and y to be in a struct, having an additional Pos struct within the objPos class only results in extra syntax (for the struct pointer) when accessing the x and y variables, as seen in the image below:

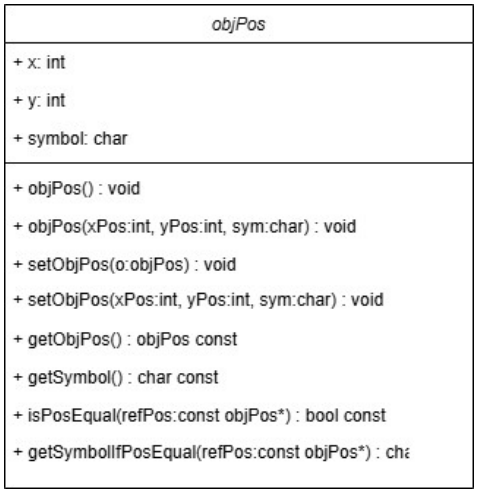
```
// Check if this position is equal to another position
bool objPos::isPosEqual(const objPos* refPos) const
{
    return (refPos->pos->x == pos->x && refPos->pos->y == pos->y);
}
```

   One can argue that since the reference to the struct is a pointer, it does not need to be instantiated and would save memory. However, there is no case in the program where an objPos object is created and the x and y coordinates are not needed. The struct would always need to be instantiated, rendering this argument invalid.

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.

Having x and y as actual fields of objPos instead of within a struct would eliminate the need of having to access the variables through the Pos pointer and would allow the program to access the variables directly. This would save computational power, albeit the savings would be insignificant on the total computational power needed to run the program. The updated UML diagram would look like the following:



Note that the objPos class does not compose of a Pos struct anymore, and x and y have become fields of objPos.