COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members golwalaz singp32

Team Members Evaluated kim600 wangs562

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 loop is very well organized and structured making it easy to follow. The get input integrates well with the game mechanics. They are all initialized at the start of the program and memory is allocated on the heap, then deleted at the end as part of the clean up routine. Although there is a slight problem with redundant reallocation, for example the mainPlayer object was initialized twice, but not deallocated properly. Each game object is descriptively named making its role easy to follow throughout the program. The way that the methods are called could be cleaner. Instead of calling getter methods multiple times, it may be easier to create a class member that stores the value of the return from the getter. For example, in the drawScreen a line like this could be used to make code run smoother objPosArrayList* foodPos = myFood->getFoodPos();. This would also make it more clear as to how objects interact within program logic.

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:
   - C++ OOD is a more organized approach, allowing for a simplistic main function where most of the complicated logic is stored in classes.
   - There is also less repetition than in procedural design needed as functions are created to handle repeated actions

   Cons:

   - When debugging it can be difficult to trace the root of the error as many classes and functions contribute to one feature of the code, and it is difficult to determine which is causing the bug.
   - C++ OOD may be more prone to memory leakage when being passed by reference into class functions, this problem is more minimal in procedural programming.

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 some comments to help understand the functionality. However, in more complicated functions in the Food and Player class, more commenting could have been used to help understand larger coding blocks with many loops and if statements. In addition to the overall logic being explained at the start of the code block, the resulting if statements and inner loops could also be commented to help a reader follow along.

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.

   Newline was deployed for proper readability in the code output. The code also includes good indentation practice for each code block. One shortcoming was the lack of whitespace in large, convoluted blocks of code. This could be fixed by adding indentations before and after inner loops/ if statements. This would make the code much easier to read.

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 game had a very smooth player experience and there was very little to critique, there was only one small bug that came up during game play. The game included a bonus feature that reduced the length of the snake by 5 characters when it hits the character 'Y'. Although when being played through, if the snake is 5 stars long and hits the 'Y', the entire snake disappears. Possible root of this error is in the movePlayer method in the player class. The logic checks to see what food was hit. If 'Y' it goes through another condition to check the length of the playerPosArrayList. If the length is less than or equal to 5 only the head of the snake is kept, otherwise 5 stars are taken off. The team should utilize print statements to print the value of playerPosArrayList at the time the condition is checked, and also look to see what the array is comprised of. They would see that hitting the 'Y' add to the listSize, not fulfilling the condition.

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.

```
FINAL SUMMARY:

DUPLICATE ERROR COUNTS:
        Error #   1:     24
        Error #   2:     21
        Error #   3:     14
        Error #   4:     14
        Error #   5:     14
        Error #   6:     14
        Error #   7:     14
        Error #   9:      2
        Error #  10:      2

SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      8 unique,   116 total uninitialized access(es)
      2 unique,     4 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)
      3 unique,     3 total,   9624 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
ERRORS IGNORED:
      7 potential error(s) (suspected false positives)
         (details: C:\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.26636.000\potential_errors.txt)
     25 unique,    26 total,   6972 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
Details: C:\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.26636.000\results.txt
```

The drmemory report shows that there are 9634 bytes of leaks. The potential root of the memory leakage is that mainPlayer is allocated twice, but only deleted once.

```
void Initialize(void)
{
    MacUILib_init();
    MacUILib_clearScreen();

    mainGameMechs = new GameMechs();
    mainPlayer = new Player(mainGameMechs, mainFoodBucket);
    mainFoodBucket = new Food(mainPlayer);
    mainPlayer = new Player(mainGameMechs, mainFoodBucket);
    mainFoodBucket->generateFood(mainGameMechs->getBoardSizeX(), mainGameMechs->getBoardSizeY());
}
```

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 do not thing the compound object design is sensible. Since the x and y coordinate variables are already stored in the objPos class, it is not necessary to further store them in a struct. This just makes it so that there is an additional step needed to access x and y.

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).

   An alternative class design would include x-coordinate and y-coordinate variables in the class freely (not inside a struct), these values would be stored in xCord and yCord. This would not have a large impact on the existing functions in the objPos class as shown in the UML diagram.

ObjPos
+ xCord: int + yCord: int + symbol: char
+ objPos () + objPos (x: int, y: int, symbol: char) + setObjPos (o: objPos) : void + setObjPos (x: int, y: int, symbol: char) : void + getObjPos () : objPos const + getSymbol () : char const + isPosEqual (refPos: const objPos*): bool const + getSymbolIfPosEqual (refPos: const objPos*): char const