COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Yash Panchal Shiv Patel

Team Members Evaluated Kamya Amaiam

Team Name Evaluating: char-group-kamya-amaiam

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 project loop is clean and well-commented. The interactions of the objects can easily be interpreted, with the names of the variables contributing to the readability. For example, the game object created using GameMechs() is assigned the name "myGM", and the player object is created using Player(myGM), which is assigned the name "myPlayer". It can easily be interpreted that the Player constructor uses the game mechanics object as an input for the constructor, giving it access to the information stored in myGM object. The implementation for adjusting speed is clear, with the values for the delays stored in an array, and accessed through the keys "q" and "e" which increment or decrement the index being read. In addition, the logic for printing the board is commented on well, and the usage of methods, and why they are being used is apparent as well. For example, in printing the board, it can be seen that the playerPos->getSize() is being used to determine how many times the loop should run, to print the player using its respective symbol (obtained using playerPos->getElement(k).symbol).

   The names of methods being used are self-explanatory, and therefore easy to follow. For example, it is clear that "int boardX = myGM->getBoardSizeX()" gets the row's value and stores it in an integer variable, and is used to track boundary conditions. A feature that should have been there is a logic for a winning condition. If a player were to grow the snake as long as the board space, then the game should be over because the player won. This can introduce bugs when approaching the end of the game, as there can be issues with memory allocation, and segmentation errors. Stemming from this, accessing the code in objPosArrayList shows that the maximum length of the snake is capped at 200, while there are 364 empty spaces on the board. If the snake reaches this length, there should be a method to increase the array list size to accommodate this in the objPosArrayList.cpp file.

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.

   C++ **Pros**

   - Code can be easier to read and modularize or change
   - Classes can be implemented elsewhere (code reusability)
   - Safer (methods and attributes reduce the risk of error)

   C++ **Cons**

   - Difficult to implement, dealing with various objects, classes, inheritance, etc.
   - Slightly slower compared to procedural C, due to various "layers" in the code

   C **Pros**

   - Easier to implement and use (when working with smaller projects)
   - Code is easier to read and is executed line by line
   - Minimal additional structure or syntax needed to be implemented
   - Direct manipulation of data is easier

   C **Cons**

   - Code in C isn't typically modular
   - Even with the use of functions, code is harder to be reused

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.

   Overall, the code does have good commenting, especially within the separate classes. The commenting conveys what the code is meant to do, in a simple, straightforward manner. It helps follow through with the code, and functionality of the code. However, in some spots, it would be nice if the commenting was more descriptive of the code itself than its functionality.
   For example, this code:

```
if (i == playerPos->getElement(k).pos->y && j == playerPos->getElement(k).pos->x) //draws player
    {
      MacUILib_printf("%c", playerPos->getElement(k).symbol);
      isPrinted = true;
      break;
    }
```

   is commented with "draws player". However, it would be better if it described what was happening in this block of code, such as "draws player position if current position matches player position", which would describe the use of comparing "i" with "playerPos->getElement(k).pos->y" better. There were some codes commented in this manner throughout the program,which would benefit from more descriptive commenting, especially if the logic is a bit complex, or is hard to read.

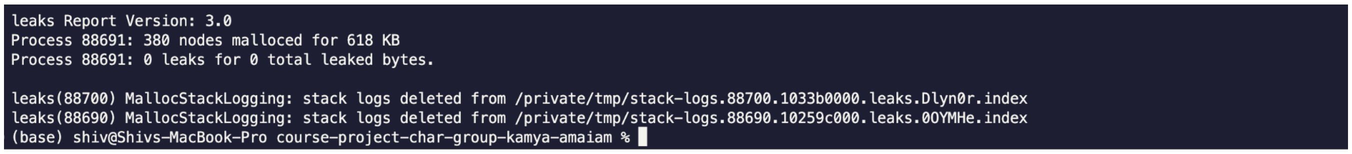
2. **[3 marks]** Does the code follow good indentation, add sensible white spaces, and deploys newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.
3. The first thing I noticed when the code was opened was how clean, and simple it looked. Though the game itself is difficult, cleverly using spaces in code, and newline characters for the terminal made it look like a simple game, and was pleasing to the eyes. In the classes, the constructors, destructors, and other methods all have a line of space between them, allowing for them to be easily distinguished, adding to the readability. In the project.cpp file, where the classes all come together, it was done in a seamless way. The header files, preprocessor directives, and global variables, and function prototypes were all placed at the top of the program, with comments and spaces between them. All of the functions were defined at the bottom of the program, underneath the main routine, with spaces used effectively between them. In addition, the indentations were clean, and there were spaces placed between lines, or chunks of code, to distinguish between logic, or lines of code that are meant to operate together. Spaces in the same line of code are also added to the readability, for example, in the project.cpp file, there is a line: "i == foodPos.pos->y && j == foodPos.pos->x". The spaces are used effectively between the operators, and variables, which improves readability. A poor counterpart to that would be "i==foodPos.pos->y&&j==foodPos.pos->x", where readability is compromised, making it seem like "y&&j" are the two statements also being evaluated.

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 offers a smooth playing experience, with easy to use controls ("WASD" for movement and "q" and "e" for speed). The keys are placed right next to each other, making it easy to play. In addition, the quitting key, "esc", is also perfect because it is self-explanatory, and is what a player might want to press without thinking to quit the game. The playing experience itself is smooth, although glitches are happening between movements. However, this is not something that they are able to control themselves, as the issue stems from the "MacUILib.h" and the IDE itself, since the terminal is being refreshed at a quick pace due to the "MacUILib_clearScreen(); " function, causing some glitchiness. The player itself moves accordingly, and is responsive to the movement keys. There is no actual flaw to be seen in the playing experience.

2. **[3 marks]** Does the Snake Game cause memory leaks? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.



```
leaks Report Version: 3.0
Process 88691: 380 nodes malloced for 618 KB
Process 88691: 0 leaks for 0 total leaked bytes.

leaks(88700) MallocStackLogging: stack logs deleted from /private/tmp/stack-logs.88700.1033b0000.leaks.Dlyn0r.index
leaks(88690) MallocStackLogging: stack logs deleted from /private/tmp/stack-logs.88690.10259c000.leaks.0OYMHe.index
(base) shiv@Shivs-MacBook-Pro course-project-char-group-kamya-amaiam %
```

Figure 1: Memory Leak Report of "char-group-kamya-amaiam" code

As seen in **Figure 1**, their code contains no memory leak. When scanning through their code, we noticed that anywhere any heap memory was allocated, they made sure to deallocate it as well. In addition, they also have the necessary destructors implemented as well, which also takes care of memory deallocation.

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?

   In our opinion, the compound object design of the **objPos** class is not sensible. We say this because it relies on dynamic memory allocation for the **Pos** structure, as the structure has only two values. The area of concern is that if not approached and coded properly, the **Pos*** (pointer) could lead to larger issues in the code, specifically memory leakage. This current approach also results in required additional implementation, specifically the *rule of six/minimum four*, requiring a *deconstructor*, *copy constructor*, etc. As **Pos** is a pretty straightforward structure, it would be more reasonable and safer to use it and implement it being value-based. To summarize, although this approach does work (as evident in our project), it overcomplicates the code and if not accommodated and implemented correctly, it could lead to major issues, such as dangling pointers or abnormal behavior, and is not a common practice or an efficient practice.

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).

In the updated design, as shown in **Diagram 1**, x and y are stored as directly accessible integer values, which allows for them to be safer, accessed quicker and more efficiently, as there is no memory allocation or deallocation/dereferencing required, and no need to implement the *rule of six/minimum four*. The rest of the class remains the same, but now the get and set methods can be used directly, and no need to involve pointers, pointing to the struct.

objPos
+ x: int
+ y: int
+ symbol: char
+ objPos()
+ objPos(xPos:int, yPos: int, sym:char)
+ setObjPos(o:objPos) void
+ setObjPos(xPos:int, yPos:int, sym:char): Void
+ getObjPos(): objPos const
+ getSymbol(): char const
+ isPosEqual(refPos: const objPos*): bool const
+ getSymbolIfPosEqual(refPos:const objPos*): char const
+ xPosIncrease(): void
+ xPosDecrease(): void
+ yPosIncrease(): void
+ yPosDecrease(): void

Diagram 1: Updated UML diagram for an alternative **objPos** class design