COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Aakash Jain (jaina78), Pasandu Sirisena (sirisenp)

Team Members Evaluated Arora_net: meloi3, yingj14

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 is well-structured with distinct phases for input, logic, drawing, and (additionally) delay, so it is easy to follow. It is also designed in a modular fashion such that responsibilities are specialized among the GameMechs, Player, and Food classes, which is good OOD practice. If there was a potentially negative feature, it would most likely be the getLastinput function in the RunLogic function as it could potentially be redundant. Another potential negative is that they are initializing variables in DrawScreen(), but I would have put that in the Initialize function because DrawScreen should just be for drawing and putting things onto the screen.
   Lines that I would move to Initialize():

```
void DrawScreen(void)
{
    MacUILib_clearScreen();

    // get boardsize
    int boardY = mechanics->getBoardSizeY();
    int boardX = mechanics->getBoardSizeX();

    //current player pos
    objPosArrayList *temp = player_ptr->getPlayerPos();
    objPos temp_player = temp->getElement(0);

    int playerx = temp_player.pos->x;
    int playery = temp_player.pos->y;

    int score = mechanics->getScore();
```

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.
   The C++ OOD approach provides better organization, scalability through future expansion (E.g., implementing the Food class later on), and data abstraction, however, there is increased complexity in the setup and memory management, and is prone to over-engineering through encapsulation (e.g., the relationship between objPos and objPosArrayList). In contrast, the C procedural approach is simpler and requires less setup, but is more difficult to scale, relies heavily on global variables, and can result in code duplication. For these reasons, the C++ OOD approach is better for larger, complex projects, while the procedural one is better for smaller, less demanding tasks.

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.
   Across all files, the code offers sufficient comments which provides insight into the purpose of the most important functions and operations to understand them efficiently. However, some of the less important functions could benefit from more documentation – for example, I am still not entirely sure the purpose of the getLastinput function, and I would improve it by providing a more detailed explanation within the Project.cpp and GameMechs files.

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 practices, uses whitespace wherever necessary, and uses newline formatting properly. The DrawScreen function is also well written and user-friendly, using newline formatting properly, but it could potentially be simplified for readability and efficiency.

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 is smooth and completely free of bugs. It should be noted that there is no way to exit the game without losing, meaning that there is no proper exit button. I would recommend that the team should add a exit key input statement in their getInput function in Project.cpp with setExitTrue().

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 were reported by Dr. Memory, indicating that the code properly handles dynamic memory allocation.
   Results: 0 bytes of leaks.

```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       5 unique,     9 total uninitialized access(es)
~~Dr.M~~       1 unique,    62 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)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~       8 potential error(s) (suspected false positives)
~~Dr.M~~          (details: C:\Users\aakas\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemor
y-cmd.exe.10768.000\potential_errors.txt)
~~Dr.M~~       4 potential leak(s) (suspected false positives)
~~Dr.M~~          (details: C:\Users\aakas\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemor
y-cmd.exe.10768.000\potential_errors.txt)
~~Dr.M~~      76 unique,   153 total,  26730 byte(s) of still-reachable allocation(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?
   The compound object design of the objPos class is sensible as the encapsulation of the x, y, and symbol variables easily enables the creation of more objects with different parameters simply using the arrow operator. Without the present implementation, it would be harder to implement the objPosArrayList and Food classes as getter and setter methods would be required, making the code longer and more complex.

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).
   For the new class design, x and y are integers and sym is a char, which replace the Pos struct. New getter and setter methods have been included to preserve encapsulation of the variables while still retaining the same function of the class.

   UML:

objPos
+ x: int + y: int + sym: char
+ objPos() + objPos(xPos: int, yPos: int, sym: char) + setObjPos(xPos: int, yPos: int, sym: char) + setObjPos(const objPos & o): void + getObjPos(): objPos + getSymbol(): char + isPosEqual(refPos: const objPos*): bool const + getSymbolIfPosEqual(refPos: const objPos*): char + getXPos(): int + getYPos(): int + setxPos(): void + setyPos(): void