COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members Chisom Marcus Chidozie, Edward Zhu

Team Members Evaluated: Ayaan Hussain, Yuvraj Singh

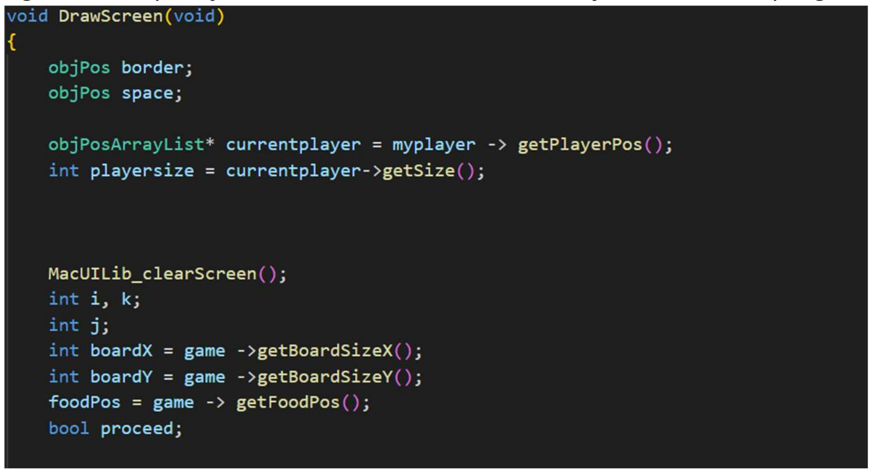
Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **25 marks**. Do not exceed 2 paragraphs per question.

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.

   Looking at the main program loop, we can very easily interpret how the objects are interacting in the main program loop. The detailed commenting on the part of the programmers add to the clarity and legibility in understanding how each object interacts in the main game loop.

   A great example of this would be the Draw Screen function in the program loop:



```
void DrawScreen(void)
{
    objPos border;
    objPos space;

    objPosArrayList* currentplayer = myplayer -> getPlayerPos();
    int playersize = currentplayer->getSize();



    MacUILib_clearScreen();
    int i, k;
    int j;
    int boardX = game ->getBoardSizeX();
    int boardY = game ->getBoardSizeY();
    foodPos = game -> getFoodPos();
    bool proceed;
```

   It is clear in showing the interactions of the different objects with each other.

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 emphasizes encapsulation by grouping related data and functionality within classes like Player, reducing accidental state modifications and enhancing code reusability. It supports modularity, abstraction, and extensibility, making it easier to add features such as animations or power-ups without impacting existing functionality. This method aligns with principles like SOLID and DRY, simplifying maintenance and ensuring clear code boundaries.

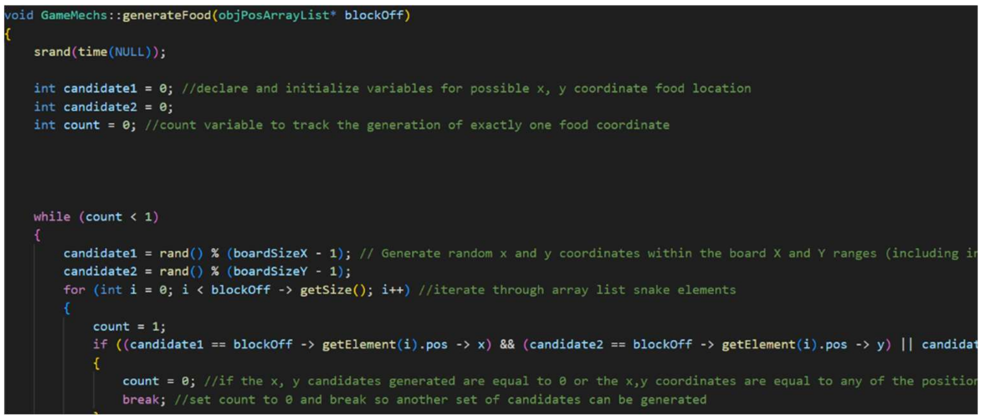
However, it introduces complexity through boilerplate code and a steeper learning curve. Additionally, features like polymorphism and dynamic memory can incur slight performance overhead, and we have found debugging class-related issues can be more challenging compared to procedural code.

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.

   From what we have observed the code seems to supply sufficient use of comments, that help clearly explain what each code block is doing.

   A great example of this would be the commenting under the Food Generation function in the GameMechs Class:



```
void GameMechs::generateFood(objPosArrayList* blockOff)
{
    srand(time(NULL));

    int candidate1 = 0; //declare and initialize variables for possible x, y coordinate food location
    int candidate2 = 0;
    int count = 0; //count variable to track the generation of exactly one food coordinate



    while (count < 1)
    {
        candidate1 = rand() % (boardSizeX - 1); // Generate random x and y coordinates within the board X and Y ranges (including i
        candidate2 = rand() % (boardSizeY - 1);
        for (int i = 0; i < blockOff -> getSize(); i++) //iterate through array list snake elements
        {
            count = 1;
            if ((candidate1 == blockOff -> getElement(i).pos -> x) && (candidate2 == blockOff -> getElement(i).pos -> y) || candidat
            {
                count = 0; //if the x, y candidates generated are equal to 0 or the x,y coordinates are equal to any of the positio
                break; //set count to 0 and break so another set of candidates can be generated
```

   As we observed the code is clearly commented explaining each sections, making us understand the code more efficiently. We don't think there are any shortcomings when it comes to the commenting.

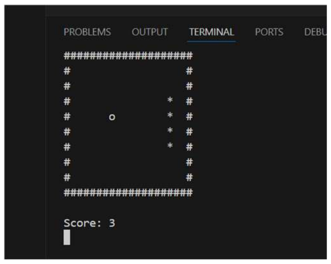
2. **[3 marks]** Does the code follow good indentation, add sensible white spaces, and deploys newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.

   We observed good indentation from the code, the team added sensible spaces an deployed the necessary newlines when formatting, this lead to a clear and consistent readability throughout the code.

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 runs remarkable smoothly, the wrap around works as intended, and the food generation as well as the body generation seems to function as intended. The Game speed is adequate, and the self body collision ends the game as intended.*
- *They used the spacebar as their exit key to terminate the program and that works as expected.*
- *The score tally also increments correctly.*
- *They appeared to use a 10x20 game board as opposed to our case where we used a 15x30 game board, we are not sure this is the right game board size.*

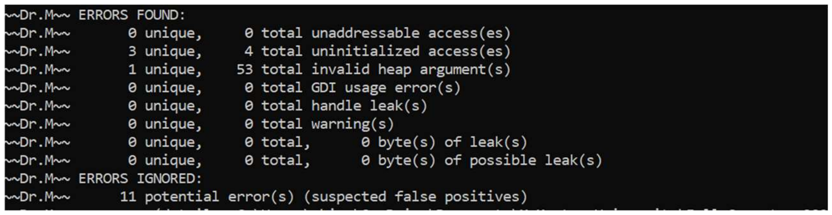


```
####################
#                  #
#                  #
#               *  #
#      o        *  #
#               *  #
#               *  #
#                  #
#                  #
####################

Score: 3
```

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.

   Their Snake game appears to have no memory leaks, after running dr memory this is the data we received:



```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       3 unique,     4 total uninitialized access(es)
~~Dr.M~~       1 unique,    53 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~~      11 potential error(s) (suspected false positives)
```

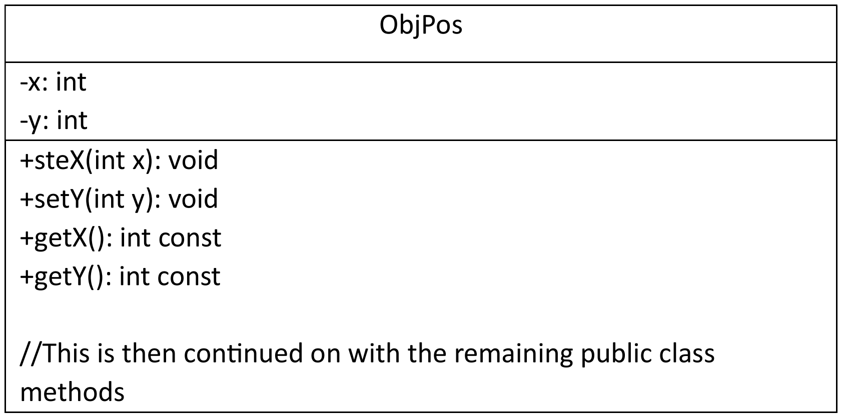
Project Reflection

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 **objPos** was a well-considered choice, as it enhances the class's modularity, making the code more adaptable and easier to maintain. This structure facilitates the seamless integration of new components, significantly improving code maintainability throughout the project. Additionally, separating dimensions helps keep the code organized, ensuring clarity and structure. Given the numerous position-related aspects of the project, this design choice ensures that programmers can efficiently access and manage dimensions as intended.*

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. <u>You are expected to facilitate the discussion with UML diagram(s).</u>



ObjPos

-x: int
-y: int

+steX(int x): void
+setY(int y): void
+getX(): int const
+getY(): int const

//This is then continued on with the remaining public class methods