COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members ____Sarah Abadir____ ____Vanessa Ishak____

Team Members Evaluated ____Stefan Ionica_____ ____Sam Vijay_____

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.

   Overall, it is clear how the objects interact with each other by looking at the main logic in the main program loop. Firstly, two pointers are declared at the start in the global scope, which point to the GameMechs class and the Player class. When these pointers are instantiated on the heap in Initialize(), the instance of the Player class takes the input of the "game" pointer, indicating that these two objects interact in the internal program logic of the Player class. Due to the appropriate names of these pointers, it is easy to follow along and understand that the "game" pointer references the methods of the GameMechs class, which deals with the mechanisms of the game, like the exit flag status, generating food onto the screen, and storing the user input. Additionally, it is clear that the player_one pointer references the Player class and is used to control the movements of the player, like in the movePlayer() and updatePlayerDir() methods, which rely on the user input stored in the GameMechs class.

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 using C++ OOD Design over C procedural design:

   - C++ OOD modularization allows for intuitive use
   - OOD provides better code readability, especially for larger coding projects, whereas C procedural coding is harder to follow along
   - OOD allows for easier implementation of changes; for instance, when implementing a new Food class to generate more "special" foods for the above and beyond feature
   - OOD provides control over which methods and members are made private or public
   - OOD is also reusable and can be built upon for other applications through composition; for instance, the objPosArrayList class was used as a member of the Player class

   Cons of C++ OOD Design:

   - OOD is more time-consuming and unnecessary if the code is not complex and is relatively short
   - C procedural design is easier to debug by using the debugger, whereas with OOD it might be more difficult to determine where the bug is coming from
   - The complexity of OOD could result in the misuse of methods

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 does a decent job of making code functionality more understandable through method names and class structures. However, there is room for improvement when it comes to adding inline comments and high-level documentation. This is especially notable in the following classes:

   - GameMechs.cpp:

     Methods such as generateFood are quite clear, but in other scenarios, the lack of comments makes it difficult to follow some of the reasoning behind the decisions. This is especially shown here

     "x = rand() % (boardSizeX - 2) + 1;

     y = rand() % (boardSizeY - 2) + 1;"

     Although, one would be able to see that this would be used to ensure the food does not interfere with the boundary, one who has not coded this game, may not know that. So, while it generates random food positions, it does not explain why it is using –2 then +1. Hence, I would add a simple comment, stating that subtracting 2 from the boarder size ensures that the range stays with the inner part of the boarder, and adding 1 ensures the values don't start at 1, thus avoiding the outermost boarder.

   - ObjPosArrayList.cpp:

     This section of the code is also well done, but could use more comments, specifically in the method insertHead. It's clear this shifts all elements one step forward to make room at the head, but a comment explaining that would help. Hence, I would alter the code with following comments to make it clearer:

     "void objPosArrayList::insertHead(objPos thisPos)
     {
       int i = listSize; // start from the current size of the list,
                         //which is the index after the last element.

     //Shift all elements one position to the right to make space at the head (index 0).
     while (i > 0)
       {
         aList[i] = aList[i - 1]; // Move the element at index (i-1) to index i.
         i--; // Move to the previous element
       }
     // Place the new element at the head of the list (index 0).
       aList[0] = thisPos;
     // Increment the size of the list to account for the newly added element.
       listSize++;
     }"

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.

For the most part the code follows good indentation, adds sensible white spaces, and deploys newline formatting for better readability. However, an issue within these aspects in the code for improper indentation and alignment can be seen in some of the if statements. This is especially evident when comparing the if statements in DrawScreen() (picture on the left) to that of movePlayer() (picture on the right). The former has clean bracket indentation, while the later follows an inconsistent format, and does not line up with the rest of the brackets.

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

    int x = 0; //loop variables
    int y = 0;

    bool exit = false;
    MacUILib_printf("Snake Game (period to exit)\n");
    for (y = 0; y < (game->getBoardSizeY()); y++)
    {

        for (x = 0; x < (game->getBoardSizeX()); x++)
        {
            if (x == (game->getBoardSizeX()) - 1 || y == (game->getBoardSizeY()) - 1 || x == 
            {
                MacUILib_printf("#"); //prints border based on the board size limits defined 
            }
            else
            {
                bool placed = false; //variable to track if we have placed anything down
                for (int i = 0; i < player_one->getPlayerPos()->getSize(); i++)
                {

                    int xCheck = player_one->getPlayerPos()->getElement(i).pos->x; //check fo
                    int yCheck = player_one->getPlayerPos()->getElement(i).pos->y; //check fo

                    if (x == xCheck && y == yCheck)
                    {
                        MacUILib_printf("%c", player_one->getPlayerPos()->getElement(i).getSy
                        placed = true;
                        break;
```

```
void Player::movePlayer()
{
    // PPA3 Finite State Machine logic
    objPos newHead = (playerPosArrayList->getHeadElement());
    if (direction == U || direction == D) {
        // Moving vertically
        int newY;
        if (direction == U) {
            newY = newHead.pos->y - 1;
        } else {
            newY = newHead.pos->y + 1;
        }
        // Wraparound logic for vertical movement
        if (newY == 0) {
            newY = mainGameMechsRef->getBoardSizeY() - 2;
        } else if (newY == mainGameMechsRef->getBoardSizeY() - 1) {
            newY = 1;
        }
        newHead = objPos(newHead.pos->x, newY, '*');
    } else if (direction == L || direction == R) {
        // Moving horizontally
        int newX;
        if (direction == L) {
            newX = newHead.pos->x - 1;
        } else {
            newX = newHead.pos->x + 1;
        }
        // Wraparound logic for horizontal movement
        if (newX == 0) {
            newX = mainGameMechsRef->getBoardSizeX() - 2;
        } else if (newX == mainGameMechsRef->getBoardSizeX() - 1) {
```

On the right, the “else” is directly next to the closed bracket, and has its own open bracket in the same line, making it difficult to follow. Moreover, new line usage is inconsistent, especially in DrawScreen(). This is seen on the left, as the for loop “for (y = 0; y < (game->getBoardSizeY()); y++)“ has a new line after it, to separate it from the next for loop. However, the for loop after it, “for (x = 0; x < (game->getBoardSizeX()); x++)”, does not use that same style to separate it from the if statement. This inconsistency is clearly seen in most of the code, particularly here. Thus, leaving an inconsistent format at times. Additionally, the white spaces are more than adequate in the code as it stands, as they are sufficiently used, and the code is easy 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 Snake Game provides a smooth and bug-free playing experience. This is evident from the seamless generation of new food after the previous one is eaten, the proper handling of snake death, and the accurate growth of the snake along with the score updates. Additionally, the wrap-around logic, which allows the snake to move across the screen borders without glitches, is incorporated effectively and functions without errors. All of these features work harmoniously to deliver a fluid and enjoyable gameplay experience.

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 seem to cause some memory leaks in the drmemory report:

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      9 unique,    72 total uninitialized access(es)
      1 unique,     2 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)
      1 unique,     1 total,   3196 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
ERRORS IGNORED:
      6 potential error(s) (suspected false positives)
         (details: C:\Users\abadi\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.42720.000\potential_errors.txt)
     25 unique,    26 total,   6972 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
Details: C:\Users\abadi\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.42720.000\results.txt
```

Upon further inspection of the code, the possible root cause of the memory leakage is in the objPosArrayList class, in the destructor method. This could be the cause since aList is an array, but the delete is missing the square brackets:

```
objPosArrayList::~objPosArrayList()
{
    delete aList;
}
```

To test this theory, the square brackets were added and drmemory was run again. Here is the drmemory report after implementing this change:

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
     10 unique,    78 total uninitialized access(es)
      0 unique,     0 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)
      0 unique,     0 total,      0 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
ERRORS IGNORED:
     22 potential error(s) (suspected false positives)
         (details: C:\Users\abadi\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.23628.000\potential_errors.txt)
     34 unique,    35 total,   8262 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
Details: C:\Users\abadi\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.23628.000\results.txt
```

Other than this minor mistake, all other memory seems to be handled properly, as now there are no bytes of leak.

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?

I do not think the compound object design of the objPos class is sensible because of the pointer to the Pos struct. The Pos struct stores very little data (just the x and y coordinates), so using a pointer adds unnecessary complexity and repetition when referencing the x and y coordinates of an objPos instance. Typically, this encapsulation is useful to build upon other applications, but for this project the Pos struct is not directly used again in any other function. Rather, it is the objPos class that is used in the other classes, like in objPosArrayList, which is further used to form the Food and Player classes. Thus, for this application, it appears that the struct is unnecessary and since objPos acts as our main building block, the x and y coordinates should be incorporated directly into the objPos class.

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.

   To improve the class design, the members that are part of the struct Pos should be added directly to the objPos class as a public member, since char symbol is already a public member as well. Below is the image of the updated UML diagram of the objPos class with the improved design. This way, there is less repetition in the code when accessing the x and y coordinates of an objPos instance. The rest of the methods would remain the same and the class would provide the same functionality.