COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Anusha Broekhuyse (400250029) and

Jenisha Thevarajah (400473218)

Team Members Evaluated: Logan and Meenaa

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

Part I: OOD Quality

1. **[6 marks]** OOD is about sensible code modularization. Looking at the header files of each object, can you easily interpret the possible behaviours of the objects involved in the program, and how they would interact with each other in the program? Comment on what you have observed, both positive and negative features.

The 3 header files included in the project are: objPosArrayList.h, Player.h and GameMechs.h. Each header file is organized and all methods have descriptive names. For example, within the objPosArrayList.h header file, I can easily understand that there are functions that insertHead, insertTail, removeHead, removeTail, getHeadElement, getTailElement and getElements. Additionally, all classes have appropriate references to other classes that they interact with. For example, in the Player.h file, there is a reference to the objPosArrayList class (`objPosArrayList* getPlayerPos();`), so that an array of object positions can be formed for the snake body.

However, there is no food class to modularize the generation of random food. If a Food class was created, its methods could be called within the Player class and proper collision detection with food items could be detected.

2. **[6 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.

Although I can interpret the program, the code can be further modularized as follows:

In the Initialize() function, all objects are initialized properly, however, the addition and initialization of a Food class would further improve the random food generation logic in the program which could be used in the main program loop.

In the RunLogic() function, there are if statements that check the keyboard input such as '\' to end the game, '+' to increase the score, 'L' to trigger snake suicide and 'F' to generate food in random locations. Although these statements demonstrate basic functionality, they are not implemented correctly within the scope of the snake game. These program functions should be redesigned and implemented in the

Player class or Game Mechanics class so that the snake moves correctly, the game exits correctly and proper collision with the food and snake body is detected.

The DrawScreen() function correctly draws the gameboard, snake and food by calling the appropriate functions. However, as collision detection with the food and snake body is not correctly implemented, these features are not drawn or displayed correctly in this function.

3. **[5 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++ OOD Pros	C++ OOD Cons
The program has better organization as functions that perform similar tasks are all grouped together in one class. For example, the Player class controlled player movement, player direction, and player collision with itself or a food item.	When implementing different classes, many individual .cpp and .h files must be generated which can be difficult to manage when the program gets large. For example, each class has two files, a .cpp file and a .h header file. In the entire program, there were five main classes that we had to work with: Food, GameMechs, objPos, objPosArrayList, and Player. This is equivalent to ten files overall.

C Procedural Design Pros	C Procedural Design Cons
All code is in one main file which makes it easier to keep track of functions and variable names within smaller programs.	When the program becomes large, it is difficult to organize all the variables and keep track of what has been declared globally and locally. Unlike C++, it is difficult to implement classes in C programming.

Part II: Code Quality

1. **[5 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 overall displays enough comments and well-chosen variable names that provide a basic/good understanding of the functionality of it. However, there are some areas where the comments could be improved and more detailed. For example, within the project.cpp file, some functions lacked comments explaining their purpose or usage. In addition, in the Run Logic function, the inputs from the user, such as '/', '+', 'L', 'F', did not provide an explanation of what the program does when those inputs are received from the user:

```
        if(myGM->getInput() == '/')
        {
            myGM->setExitTrue();
        }
        // For Score
        if(myGM->getInput() == '+')
        {
            myGM->incrementScore();
        }
        // For Suicide
        if(myGM->getInput() == 'L')
        {
            myGM->setLoseFlag();
            myGM->setExitTrue();
        }
        // 3.1 logic for food generation
        if(myGM->getInput() == 'F')
        {
            myGM->generateFood(tempPos);
        }
        myGM->clearInput();
```

This portion of the code offers concise 2–3-word comments describing the outcomes, yet it lacks commentary on the overall functionality of the function.

The Draw Screen function contains complex features within it, therefore more comments within this function would help clarify the purpose of each loop and how it achieves displaying the characters on the screen. Each loop could have had comment specifying what is being looped through.

```
    }
    for (int i = 0; i < bY; i++)
    {
        for (int j = 0; j < bX; j++)
        {
            // Iterate through player list
            for(int k = 0; k < playerBody->getSize(); k++)
            {
                playerBody->getElement(tempBody, k);
                // Drawing player
                if (i == tempBody.y && j == tempBody.x)
                {
                    board[i][j] = tempBody.symbol;
                }
            }
```

Providing a description of the memory allocation purpose within the draw screen's Initialize function would have enhanced comprehension of the code's functionality.

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

    myGM = new GameMechs();
    myPlayer = new Player(myGM);

    objPos tempPos{1, 1, 'o'};
    myGM->generateFood(tempPos);
}
```

Also, within the Player.cpp file, it would be beneficial to elaborate on which specific on what specific action are being don't and why, as this file contains the movement and action of the player. Within the movePlayer() function, the code has some comments that offer insights into the functionality of certain sections and functions. However, there are opportunities for improvement in terms of clarity.

```
void Player::movePlayer()
{
    // PPA3 Finite State Machine logic
    objPos currHead; // holding current head pos info
    playerPosList->getHeadElement(currHead);

    switch (myDir)
    {
        case UP:
            currHead.y --;
            if (currHead.y <= 0)
            {
                currHead.y = mainGameMechsRef->getBoardSizeY() - 2;
            }
            break;
        case DOWN:
            currHead.y ++;
            if (currHead.y >= mainGameMechsRef->getBoardSizeY()-1)
            {
                currHead.y = 1;
            }
            break;
```

In the constructor, additional comments could be beneficial to explain the purpose of creating a snake with six segments.

```
    playerPosList->insertHead(tempPos);
    playerPosList->insertHead(tempPos);
    playerPosList->insertHead(tempPos);
    playerPosList->insertHead(tempPos);
    playerPosList->insertHead(tempPos);
    playerPosList->insertHead(tempPos);
```

Comments in the header file, such as getPlayerPos(), could be refined for enhanced clarity. Finally, including brief method descriptions at the beginning of each function would allow a quicker understanding of their roles in the overall functionality.

```
public:
    enum Dir {UP, DOWN, LEFT, RIGHT, STOP};  // This

    Player(GameMechs* thisGMRef);
    ~Player();

    objPosArrayList* getPlayerPos();
    void updatePlayerDir();
    void movePlayer();

private:
    // Reference to objPosArrayList
    objPosArrayList *playerPosList;
    enum Dir myDir;

    // Reference to the Main Game Mechanisms
    GameMechs* mainGameMechsRef;
```

These enhancements would contribute to a more self-documenting and comprehensible codebase.

2. **[4 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.

This code demonstrates generally good practices in terms of indentation, sensible use of white spaces, and newline formatting, contributing to its readability. The functions are appropriately indented, and there is consistent spacing within expressions.

For example, within the code provided below, it displays a well-organized structure exemplifying effective spacing techniques that enhance code readability:

```
void RunLogic(void)
{
// 3.1 logic for food generation
objPos tempPos{1, 1, 'o'};

// Player control
myPlayer->updatePlayerDir();
myPlayer->movePlayer();


// To end game
if(myGM->getInput() == '/')
{
myGM->setExitTrue();
}
// For Score
if(myGM->getInput() == '+')
{
myGM->incrementScore();
}
```

However, for consistency, it would be better to standardize the brace style throughout the code, ensuring that opening braces either appear on the same line or the next line consistently across functions.

Part III: Quick Functional Evaluation

1. **[8 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 a technical user feedback)

The generateFood()method checks for overlap only with the head of the snake (blockOff). It does not consider the entire snake's body because playerPosList was not passed through as a parameter and within the generatefood() . This could lead to situations where the generated food may overlap with the snake's body, affecting gameplay as they were not able to loop through the playerPosList because the function did not take in that value. Debugging Approach; Modifying the generateFood() method to consider the entire snake's body during food generation would resolve this issue.

In addition, in the movePlayer() function, there is an absence of the collision logic to check whether the newly positioned head overlaps with the objPos of the food. This absence can lead to issues in the game's functionality and the players experience in the game. Due to the lack of collision checking with the food, the movePlayer function does not include logic to check whether the newly positioned head overlaps with the position of the food. Without collision checking, the game may not handle scenarios where the snake's head collides with the food, impacting the expected game flow. A debugging approach that would be incorporating collision checking logic within the movePlayer function.

2. **[6 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause of the memory leakage.

As shown by the memory report below, there is no memory leakage. Additionally, as seen in the code, when a new instance was created on the heap using the "new" keyword, it was correctly deallocated on the heap using the keyword, delete, as follows:

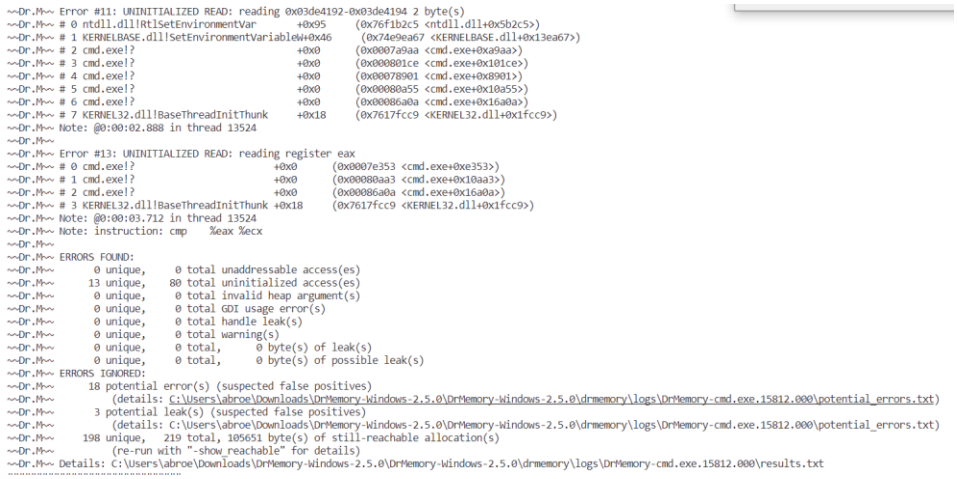
```
void Initialize(void)
{
    MacUILib_init();
    MacUILib_clearScreen();

    myGM = new GameMechs();
    myPlayer = new Player(myGM);

    objPos tempPos{1, 1, 'o'};
    myGM->generateFood(tempPos);
}
```

```
void CleanUp(void)
{
    MacUILib_clearScreen();
    if(myGM->getLoseFlagStatus() == true)
    {
        MacUILib_printf("LOSER");
    }

    delete myGM;
    delete myPlayer;
    MacUILib_uninit();
}
```



```
~~Dr.M~~ Error #11: UNINITIALIZED READ: reading 0x03de4192-0x03de4194 2 byte(s)
~~Dr.M~~ # 0 ntdll.dll!RtlSetEnvironmentVar       +0x95     (0x76f1b2c5 <ntdll.dll+0x5b2c5>)
~~Dr.M~~ # 1 KERNELBASE.dll!SetEnvironmentVariableW+0x46    (0x74e9ea67 <KERNELBASE.dll+0x13ea67>)
~~Dr.M~~ # 2 cmd.exe!?                            +0x0      (0x0007a9aa <cmd.exe+0xa9aa>)
~~Dr.M~~ # 3 cmd.exe!?                            +0x0      (0x000801ce <cmd.exe+0x101ce>)
~~Dr.M~~ # 4 cmd.exe!?                            +0x0      (0x00078901 <cmd.exe+0x78901>)
~~Dr.M~~ # 5 cmd.exe!?                            +0x0      (0x00080a55 <cmd.exe+0x10a55>)
~~Dr.M~~ # 6 cmd.exe!?                            +0x0      (0x00086a0a <cmd.exe+0x16a0a>)
~~Dr.M~~ # 7 KERNEL32.dll!BaseThreadInitThunk     +0x18     (0x7617fcc9 <KERNEL32.dll+0x1fcc9>)
~~Dr.M~~ Note: @0:00:02.888 in thread 13524
~~Dr.M~~
~~Dr.M~~ Error #13: UNINITIALIZED READ: reading register eax
~~Dr.M~~ # 0 cmd.exe!?                           +0x0      (0x0007e353 <cmd.exe+0xe353>)
~~Dr.M~~ # 1 cmd.exe!?                           +0x0      (0x00080aa3 <cmd.exe+0x10aa3>)
~~Dr.M~~ # 2 cmd.exe!?                           +0x0      (0x00086a0a <cmd.exe+0x16a0a>)
~~Dr.M~~ # 3 KERNEL32.dll!BaseThreadInitThunk +0x18     (0x7617fcc9 <KERNEL32.dll+0x1fcc9>)
~~Dr.M~~ Note: @0:00:03.712 in thread 13524
~~Dr.M~~ Note: instruction: cmp    %eax %ecx
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~      13 unique,    80 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 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~~      18 potential error(s) (suspected false positives)
~~Dr.M~~         (details: C:\Users\abroe\Downloads\DrMemory-Windows-2.5.0\DrMemory-Windows-2.5.0\drmemory\logs\DrMemory-cmd.exe.15812.000\potential_errors.txt)
~~Dr.M~~       3 potential leak(s) (suspected false positives)
~~Dr.M~~         (details: C:\Users\abroe\Downloads\DrMemory-Windows-2.5.0\DrMemory-Windows-2.5.0\drmemory\logs\DrMemory-cmd.exe.15812.000\potential_errors.txt)
~~Dr.M~~     198 unique,   219 total, 105651 byte(s) of still-reachable allocation(s)
~~Dr.M~~         (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: C:\Users\abroe\Downloads\DrMemory-Windows-2.5.0\DrMemory-Windows-2.5.0\drmemory\logs\DrMemory-cmd.exe.15812.000\results.txt
```

Part IV: Your Own Collaboration Experience (Ungraded)

1. Tell us about your experience in your first collaborated software development through this project – what was working and what wasn't. If you are a one-person team, tell us what you think may work better if you had a second collaborator working with you.

Overall, it was a positive experience; it was helpful working with someone as it made debugging and brainstorming ideas throughout the process easier.