COMPENG 2SH4 Project – Peer Evaluation

Your Team Members ____Li Xintong_____Xu Xikai_____

Team Members Evaluated _____Marwan__ ____Makarios_____

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

Part I: OOD Quality

1. **[5 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.

   Firstly, the program of this group strictly follows the OOD format, subdividing the classes very finely. Among them, objPos is the most basic level, and an objarraylist is built, followed by the construction of games, players, and food. For the player part, there are game mechs pointers parts that intersect with other parts, including detection failure, setting failure, score increase, and avoiding game boundaries. The advantage of this approach is high integration, and most of the program's behavior is included in the integrated class, which has good confidentiality. However, when directly executing operations, it is easily limited by timing. Secondly, food generation. This group uses a conservative approach and only creates one food, making memory recycling relatively simple. Food generation is directly affected by the position of the snake body and the border, which is somewhat less prone to memory leakage and easy to track. However, due to the high integration of the player, the efficiency of execution may be lower.

```
void getPlayerPos(objPosArrayList &returnPos);
void updatePlayerDir();
void movePlayer(int sizeX, int sizeY);
void setFoodObj(Food currFoodObj);
void getFoodPos(objPos &currFoodPos);
```

   Note: The class contains input and output at the list class level, which may reduce IO performance

```
Player(GameMechs *thisGMRef); // Constructor
~Player();
```

```
    3. Set the player direction to STOP
    */

    int boardSizeX = thisGMRef->getBoardSizeX();
    int boardSizeY = thisGMRef->getBoardSizeY();

    mainGameMechsRef = thisGMRef;
    myDir = STOP;

    objPos tempPos;
    tempPos.setObjPos(boardSizeX / 2, boardSizeY / 2, '@');
```

```
        foodObj.generateFood(*playerPosList, sizeX, sizeY);
        mainGameMechsRef->incrementScore();

    }
    else
    {
        // new current head should be inserted to the head of the list
        playerPosList->insertHead(curr_head);

        // then remove the tail of the list
        playerPosList->removeTail();
    }
```

```
void Player::checkSelfColision()
{
    objPos snakeHead, snakeBody;
    playerPosList->getHeadElement(snakeHead);
    for(int i = 1; i < playerPosList->getSize(); i++)
    {
        playerPosList->getElement(snakeBody, i);
        if(snakeHead.getX() == snakeBody.getX() && snakeHead.getY() == snakeBody.getY())
            mainGameMechsRef->setLoseFlag();
    }
```

```
    void generateFood(objPosArrayList blockedPositions, int sizeX, int sizeY);
    void getFoodPos(objPos &returnPos);
```

The image output process of the program is quite complex, including border data extraction, new bookstore data and snake body data (using the class of food), temporary judgments, and judgmental border output. The advantage of the output process is that it is easy to modify and flexible, but excessive creation and neglect of some objpos and list functions make the output process appear quite cumbersome, The xy value obtained from scanning the line can be interactively output as a bool value in the list class to determine whether to print, reducing program complexity.

```
drawn = false;

// draw the player body
for (int k = 0; k < curr_player_body.getSize(); k++)
{
    curr_player_body.getElement(temp_body, k);
    if (temp_body.getX() == j && temp_body.getY() == i)
    {
        MacUILib_printf("%c", temp_body.getSymbol());
        drawn = true;
        break;
    }
}

// skip to next iteration if the player body is drawn
if (drawn)
    continue;

if (i == 0 || i == boardSizeY - 1 || j == 0 || j == boardSizeX - 1)
{
    MacUILib_printf("#"); // Set border to '#'
}
// draw inner spaces
else if (i == curr_food_pos.getY() && j == curr_food_pos.getX())
{
    MacUILib_printf("%c", curr_food_pos.getSymbol()); // Print food
}
else
{
    MacUILib_printf(" "); // Set inner spaces to ' '
}
```

[5 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 includes the interaction between borders, snakes and food, and image output.

```
void RunLogic(void)
{
    // get the user input
    char userInput = gameMechs->getInput();
```

This section assigns the input process to the runtime, which I believe will reduce the efficiency of memory usage. Writing data immediately will reduce memory usage time, but timing issues can be reduced during runtime.

```
// update the player direction based on the user input
player->updatePlayerDir();
player->movePlayer(gameMechs->getBoardSizeX(), gameMechs->getBoardSizeY());

// After the player moves, update the player position in the curr_player_body objPosArrayList
curr_player_body.copy(player->getPlayerPos());
player->checkSelfColision();
```

Due to excellent logical design, collisions, bonus points, and judgments will be hidden, making the main program appear to have very little interaction. However, constantly writing back to the position of the snake will continuously compare the position between the snake and food. Therefore, the program integration is good, but it will affect the debugging ability of the program. A large number of functions should interact in runlogic, and class interaction will affect the independence between each module, Contrary to the purpose of the black box model designed by OOD.

The part that I don't understand is that the player needs to constantly compare with the border because the data is all in the GM. High intensity IO will reduce program performance, so it is better to directly use the data in the pointer.

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 program in PPA3 uses a flat design, which makes it easy to understand and debug, and the execution timing is easy to track. The disadvantage is poor sealing, and when the program is large in scale, resource waste is severe. When using OOD design, due to modular design, the file size is huge and difficult to understand, making it difficult to debug and track timing. However, OOD design has good security and can design huge projects while also having good scalability, If the main program wants to add features, simply add a new header file.

Part II: Code Quality

1. **[4 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.

In the code, most of the functions and constructor contains enough comments to describe what the function do, which make me understand the code easily. Especially for the “insertHead”,“insertTail”,“removeHead”,“removeTail”, there are sufficient comments to describe every step of the function to help us understanding. However, some of the functions like “checkSelfCollision” do not have enough code. Maybe they could add some code like “get snakeHead postion from the playerposList, and compare the head position with the body position which is got by ‘getElement’ function in the ‘for’ loop.”

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.

In their code, for every condition and loops, all the statements and brackets follow the correct indentation, which play a role in reading and identifying the boundary of the loop or condition. Whitespaces is used between the operators and operands. Besides, newline formatting is also used to divide a function into different blocks of code, which organizes the function efficiently.

Part III: Quick Functional Evaluation

1. **[6 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)

In the Food.cpp, at line 36, the condition of the regeneration of the food, "&&" should be used to replace the "||", because the food can be generated in the board at the position without overlapping the snake rather than the position that at the same rows or columns of the snake body.

```
    // Generate random candidate coordinates until an unblocked position is found
    do
    {
        candidateX = rand() % (sizeX - 2) + 1;
        candidateY = rand() % (sizeY - 2) + 1;          //with in the same coordinate with the snake
    } while (xCoordinateBitVector[candidateX - 1] == 1 || yCoordinateBitVector[candidateY - 1] == 1)

    // Set the food position to the unblocked position
    foodPos.setObjPos(candidateX, candidateY, 'O');


void Food::getFoodPos(objPos &returnPos)

    returnPos.setObjPos(foodPos.getX(), foodPos.getY(), 'O');
```

```
##############################
#@@                           #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@                            #
#@@@@                         #
##############################
score is 16
```

//NO LONGER GENERATED ANY FOODS

Overall, the program has good stability, runs smoothly, and generates accurate food. Good jobs!

2. **[4 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.

Temporary, we do not find the memory leak in their Snake Game code.

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.

   Collaborative programming requires good communication skills. In this assignment, we found it difficult to interact with each other's ideas. However, due to the black box nature of OOD, we still struggled to complete the assignment.