

COMPENG 2SH4 Project – Peer Evaluation

Your Team Members _____Yuan Ding_____Warun Gumarawell_____

Team Members Evaluated _____Arabella Paet_____Raymond Pham_____

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.

For Food.h(included are objPos.h, objPosArrayList.h, GameMechs.h, MacUILib.h):

Yes, it is easy to interpret the food class. From looking at the header file, it pretty clear that this class is responsible for the generation of the food and its associated functions in the game. It is organized well and it very clear to read. Constructors, deconstructors, getter and other methods are clearly named and grouped and also are well-commented. However, it would be ideal if the last 2 methods were also commented as it is not very straightforward to understand just from the method name. We can also see that objPos.h, objPosArrayList.h, GameMechs.h, and MacUILib.h are included in the code as they are interacted with in the code.

For GameMechs.h(included are objPos.h, objPosArrayList.h, MacUILib.h):

Yes, it is easy to interpret the Game Mechs class. This is because it is well-commented, and the code is laid out in a very organized manner allowing us to easily understand what certain parts of the code are doing without knowing the actual code. The member functions and variables are laid out in a way that groups them depending on what they do. In the code, the Getter methods are commented and kept separate from the setter methods, which allows for easy reading. Most importantly, the method names are created in a way that exactly tells you what it will do, and thus you can understand what it will do without knowing the actual code. It should also be noted that comments are kept to a minimum, since the method names are self-explanatory, and as a result, the code is not bloated with unnecessary comments. Just from looking at the header file, it is pretty clear that this class is responsible for the rules and mechanics of the game, the general function of the game.

For player.h(included are GameMechs.h, objPos.h, objPosArrayList.h, Food.h)

Yes, it is easy to interpret the player class. From looking at the header file, it is pretty clear that this class is responsible for how the player moves around and interacts with the game components. Looking at the constructor, we can see that it interacts with the food class and gamemechs class since it is passed into the constructor for the class. We can also see that this class interacts a lot with objects from the objPos

class. For many of the methods, objPos head is passed. For example, for increasePlayerLength function, it takes objPos head as parameter and increases the size of the list if the snake consumes a food. The code here is also well commented and grouped, making it very easy to read and understand. OOD principles are used very well here.

For objPosArrayList.h(included are objPos.h):

Yes, it is easy to interpret the objArrayList class. From looking at the header file and its methods, it is pretty clear that this class is used for the body of the snake player. Given the method and parameters, we can see that this class works with the objPos class. The code is grouped accordingly and well-commented. Commenting is kept to a minimum which is good as the method names and parameters are self-explanatory and as a result, the code is not bloated with unnecessary comments. However, it may have been beneficial to add some comments that explain what is being done with the parameter of the method.

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.

Yes, you can easily interpret how the objects interact with each other in the programming logic by the code and it does so in a clear, understandable and concise manner. The main logic, RunLogic(), is clear and has only 3 lines of code, keeping it very concise. These 3 lines of code are very easy to understand given their names and comments. As a result, you can easily interpret how the code will behave. The first line of code uses ptrPlayer, a player object, to call a method within the player class, which is updatePlayerDir(). Here, given the name of the method and comments, we can clearly see that this will take input from the user and update the player direction accordingly. The second line, similar to the first line, now calls the movePlayer() method from the player class, which moves the player given the direction from the previous line of code, while also heading to the wraparound logic. The third line of code ptrGameMechs object calls the clearInput() method. Given the name of the method, we can see that this method will clear input user input from the stream so that . As a result of the main logic being concise, well-commented and named well, it is very easy for someone to interpret the code and how it interacts and behaves. Perhaps one thing that can be added is comments for the clearInput line, however, that is a small detail since the method is relatively self explanatory for the most part.

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.

Pros:

- Modularity
 - o With classes and objects, you can break down the project into smaller, more manageable systems/sub-tasks

- Classes will have its own set of variables and specific methods to complete a certain task, rather than having functions and variables being declared together
- Reusability
 - The classes can easily be integrated into other projects
 - You can build upon classes and as a result, reduce redundancy and speed up the development process in later projects
 - With inheritance, you do not need to repeatedly write the same attributes of class
- Readability/Organization
 - Code is structured in a clear manner
 - Classes are a natural way of representing objects, thus more intuitive/readable
 - Having classes, methods and attribute that are named well can reduce the amount of comments and make it easier to understand/read
- Ease of Debugging/Troubleshooting
 - You can debug one class at a time
 - You can trace the bug to a single class since you know the function/behavior of the class

Cons:

- Learning Curve
 - OOD approach is more challenging than the traditional procedural approach
 - Requires much more time and effort to implement
- Organization Difficulty
 - OOD does make the code more organized, however, it does introduce more files in the project
 - Need to keep track of a lot of different files, which may be difficult
- Private/Public Access
 - In contrast to procedural design, we must ensure that private and public data members are set correctly
 - Must properly control visibility of class members
- Performance
 - OOD may result in higher use of resources compared to procedural
 - Creating objects and such may affect runtime performance
 - Especially if the device/hardware is resource limited

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 has a great number of comments to help understand the code functionality more efficiently. Comments are generally used to group different types of methods, which ideal for organization and better readability. For example, getters, setters, constructors and destructors are all grouped and commented, and this makes it much easier to read and understand the code. Moreover, when the function of a certain block of code is difficult to understand just by the code, comments are used to help.

An example of this is can be seen in Project.cpp.

```
//If an element of the snakes body has been printed onto the board, skip the rest of the actions below and go to the next coordinate of the board
if(drawBody)
{
    continue;
}
```

Here, we can see block of code that is pretty simple, however, its purpose/functionality may not be so obvious. The comments here ensure that the functionality of this block of code is specified. Having well-commented code is very useful and allows for quicker readability of code as you do not have to go back and forth between different pieces of code.

However, there are sometimes where commenting is redundant.

```
void GetInput(void)
{
    ptrGameMechs->getInput();           //takes input from the user
}

void RunLogic(void)
{
    ptrPlayer->updatePlayerDir();        //takes input from user for the player objects direction
    ptrPlayer->movePlayer();             //move player object and head wraparound logic
    ptrGameMechs->clearInput();
}

void DrawScreen(void)
{
    MacUILib_clearScreen();
    objPos board;                       //object for each coordinate on the board
    objPos tempObj;
    objPosArrayList* playerBody;
    objPosArrayList* foodBucket;
    bool drawBody;                      //indicator for if an element from the snakes body has been printed or not
    bool isFood;
    int foodIndex, bodyIndex, column, row;
```

Here, we can see that the comment for the getInput() method is not needed as it is self-explanatory given the method name.

However, there are some times where commenting would be beneficial. For example:

```
//Iterate through each food element inside the list (foodBucket)
for(foodIndex = 0; foodIndex < foodBucket->getSize(); foodIndex++)
{
    foodBucket->getElement(tempObj, foodIndex);
    if(tempObj.isPosEqual(&board))
    {
        cout << tempObj.getSymbol();
        isFood = true;
        break;
    }
}
```

Here, we can see that it may have been beneficial to include a few more comments for this block of code. The comment here mentions that it iterates through each food element inside the list, however, it does not specify what is exactly doing with it.

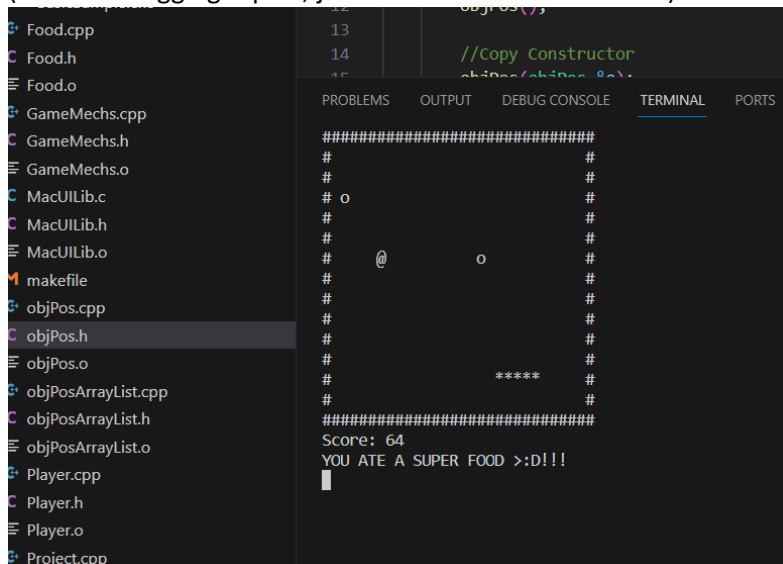
Overall, for the most part, commenting and self-documenting has been very well done.

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.

Yes, the code follows good indentation, adds sensible whitespaces and deploys newline formatting for better readability. Code within functions are indented for clear readability. For loops, while loops, and if statements are structured and indented well, making it very clear to read. There are times when comments are on one singular line and are very long. This is not a huge deal, but it does affect readability. It may have been beneficial to break up the comments into smaller parts, and place them at more precise locations, instead of just above the main code block. White spaces are also used well and can be seen in the function parameters, and conditions within the for loop, while loop and if statement. Having the white space allows for better readability as it is more spaced out. In the draw screen, new line character is also used so that the user can better read the text within the game.

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)



```
#####
#                                     #
#                                     #
# o                                 #
#                                     #
# @          o                     #
#                                     #
#                                     #
#                                     #
#                                     #
#                                     #
#                                     #
#                                     #
#                                     #
#                                     #
#                                     #
#                                     #
#####
Score: 64
YOU ATE A SUPER FOOD >:D!!!
█
```

The entire snake game has no bugs, offering a smooth, bug-free playing experience. After exchanging ideas with many other groups, I found that in many groups' games, there is often frequent flashing. However, compared to most snake games I've seen, the flashing and delay in

this group's game are relatively good. They have even designed a very unique gameplay by introducing "super food" (@), making the entire game seem more special.

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.

```
leaks Report Version: 3.0
Process 3291: 281 nodes malloced for 368 KB
Process 3291: 0 leaks for 0 total leaked bytes.
```

The answer is NO. The snake game developed by this team does not result in any memory leaks. They initialized an instance of their game mechanics Class and an instance of their play class on the heap, and both were appropriately deleted during the clean-up routine, ensuring no memory leaks occurred. In addition, destructors were also used in classes to ensure that anything created on the heap is also deleted when the program is over.

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.

The entire project involved a significant amount of coding work, and it was a very challenging project for us. Our team's code did not meet the requirements of 2SH4 because we encountered some issues in the final part that we couldn't resolve. Even though our overall code was logically sound, certain bugs, which we couldn't pinpoint, prevented our project from meeting expectations. If given another chance, we would aim for more communication within the team to detect issues promptly and reach out to the TA or even the professor immediately for assistance. This would allow us more time to address these bugs. Therefore, we believe this software development effort was not ideal overall, but next time, we will pay more attention to communication among team members and seek timely advice from the TA.