COMPENG 2SH4 Project – Peer Evaluation

Your Team Members _____Jonathan_____ _____Salih_____

Team Members Evaluated ____Hamza_____ _____Umar_____

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.

   Looking at the header files of each object, it is quite easy to tell what the possible behaviours of the objects will be, and how they will interact with each-other in the program. For example, when looking at the food class, it is easy to understand that it is responsible for generating the food in the main program. Moreover, since it has references to the object position array list, I can infer that it interacts with the player by making sure that food is not generated where the player is. Another example is when looking at the player header file, based on the reference made to the GameMechs file and Food file as well as the presence of an enumeration that controls direction, it can be inferred that the player file is responsible for processing player input, updating the players state, as well as checking for events such as eating food and collisions.

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.

   Looking at the main program loop it is easy to understand how the objects interact with each other. We see each object being defined and initialized in their appropriate locations. We see that the GetInput() function set the user input in the GameMechs object. And in run logic we see that the game exits if escape key is pressed and if the food is consumed it increases the player length and generates food. The run logic also calls functions to update the direction of the player as well as move the player. The input is then cleared. In draw screen the food is generated. The food, each part of the player and the game board are all printed. The clean up works properly and accounts for when the player runs into them self and for when the player exits the game, and it deallocates the memory for gameMechs the player and the food objects. The flow is very straightforward and the comments help explain what each portion of the code is doing.

   There are some suggestions, however. For instance, in the RunLogic() function, I would recommend using the get input in the game mechanics to set the condition for if the player presses ESC as opposed to in the Project.cpp. I also recommend moving the player consumption and growth condition into the Player object as well. The GetInput() function in project.cpp can be empty because it doesn't do anything since we call the

GameMechs version of getInput() in updatePlayerDir() in the player object Player. The memory also doesn't need to be deallocated in clean up if the appropriate destructors are implemented in the relevant classes.

Overall the main function is easy to understand and how the objects interact make sense however the developers should try to keep certain functions in their relevant objects and try to avoid cross over into Project.cpp for better readability and a more efficient OOD approach.

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:

1. OOD approach is more readable. When we see specific functions being used, we know what part of the game it is relevant to whether it is the game mechanics the food the player or the other parts of the game.
2. It can be easier to debug. Since each object is responsible for itself bugs are easier to isolate whereas in procedural design bugs can occur anywhere in the program and can be harder to spot.
3. The programmer doesn't have to spend much time trying to find specific parts of the code. They can just go to the appropriate file and make edits as necessary whereas in procedural design approach the programmer would need take time to scroll to the appropriate location.

Cons:

1. One con is having to manage multiple tabs. When following an OOD approach there are multiple files that handle each extension/subsection of the program. This results in the programmer having to juggle through all the files often having to go between tabs.
2. Another con is that it is more difficult to program in OOD approach compared to the procedural design because it is spread out across multiple files. It can get confusing to manage each object.

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.

For the most part, the code offers sufficient comments. Within the header files, there were minimal comments, but I find that to be ok since most of it was self explanatory, and whatever may not have been did have a comment beside it explaining the functionality of the line. In the main code it is very well documented using comments. The code is fairly concise and provides comments for any pointer references that are made which makes it easy to understand how different objects interact with each other.

Moreover, most of the logic such as loops also had a comment explaining what the function was. One issue that was present in the main program was the lack of comments on one specific section of the main program provided below.

```
for (k = 0; k < PlayerBody->getSize(); k++)
{
    PlayerBody->getElement(tempBody, k);
    if (tempBody.y == i && tempBody.x == j)
    {
        MacUILib_printf("%c", tempBody.getSymbol());
        directionString = myPlayer->getDirectionString();
        drawn = true;
        break;
    }
}
```

This was difficult to read as there were not any comments explaining what it was doing, which could be problematic when trying to look at the code after some time. Additionally, in the other .cpp files such as objPos.cpp there are very minimal comments which make it difficult to understand the purpose of each function.

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.

Throughout the main program of the code, proper indentation practices were followed. Within the loops everything is properly indented inside the brackets, making it easy to understand how the loop functions, and each loop is formatted the same way making the code cleanly written. The code also employs sensible white spacing making it easy to read and understand conditions within loops and print statements. Newline formatting was also properly deployed throughout the code, specifically the main program. There were newlines inserted between function definitions and between if statements and loops, which helped to distinguish where one section ended and another one began, increasing the overall readability of the code.

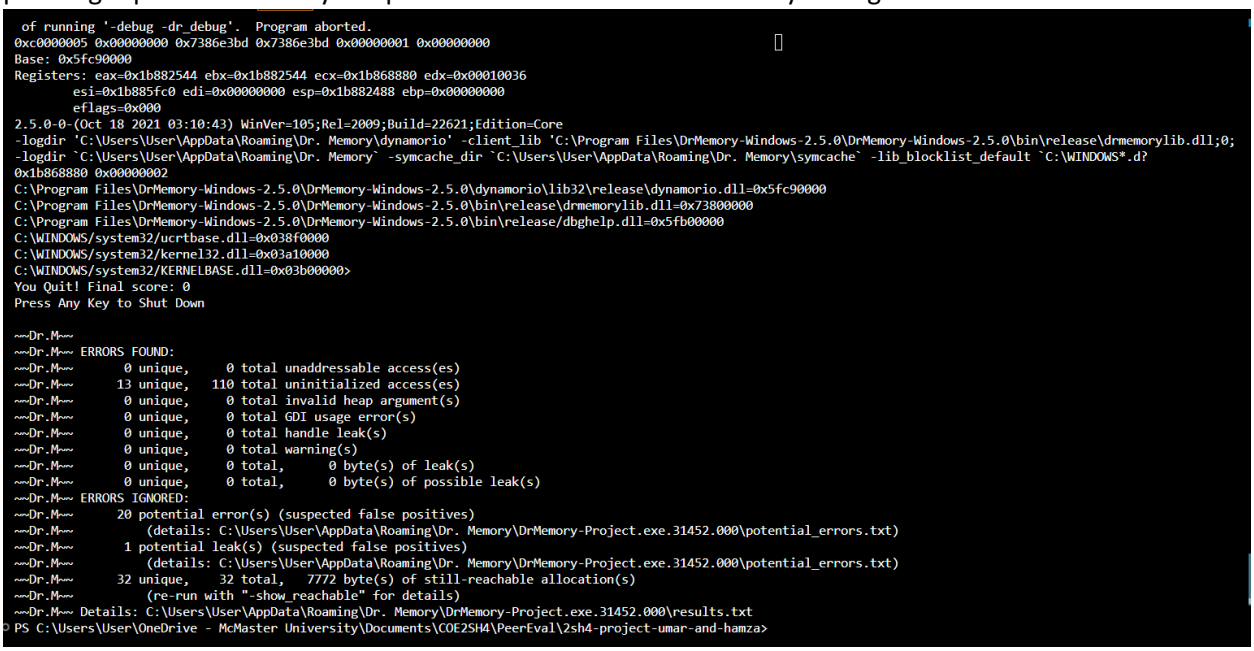
Part III: Quick Functional Evaluation

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 Snake game is bug free and runs smoothly. All portions including the food generation, the border, the wraparound, the snake, the growth by consumption, and the collision all work as intended. The score works as intended and so does the game over

and game quit screens.

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.



Based on the Dr.memory report, there are 0 bytes of memory leak. This is because they correctly deallocated memory in their cleanup routine as shown below.

```
// deallocates memory occupied by the objects on the heap.
delete gameMechs;
delete myPlayer;
delete food;
```

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, collaborating on software development was an interesting experience. Previously, the only experience we had on collaborating on software was our 1P13 projects, but it was different as we did not use GitHub. We found this project from 2SH4 to be a valuable experience as it something that will likely translate into the workforce.