COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Michael Mota and Riley Chai
Team Members Evaluated	Asheel Dowd and Simone Tabile

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, appropriate naming of classes, functions, and variables allow for a high-level overview of the object and its intended purpose. Order within class definition is correct and comprehensive (eg. Constructor -> destructor -> public etc.) It is also possible to see references to other classes and how they will work together. However, some libraries are included and then not used, which can lead to confusion and a larger file than necessary. For example, MacUILib.h is included in GameMechs.h and then never used.

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.

    - It is relatively easy to interpret how the objects interact with each other. This is because objects all interact with a purpose. My only suggestion is to change the variable names for the temporary objPos objects within the drawscreen function. A temporary object is created for the food position and the player positions. The names are temp1 and temp2, and although the comments make it clear which is which, it is still probably not a best practice. Maybe instead name them playerTemp and foodTemp?

```
objPos temp2; //food temp
objPos temp1; //player temp
temp2 = objPos();
temp1 = objPos();
```

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.

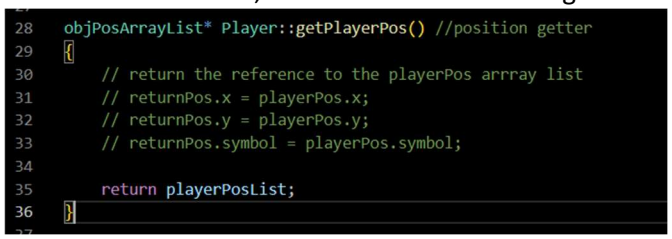
    - The advantages of using an OOD approach for the project is that it allows for greater modularity and facilitates collaboration since team members can work on different files at

the same time without interference. It also allows for chunks of code to be reused and changes can be made in one spot when moving between iterations. Some cons of the OOD approach are that it is more complicated to setup and for simple projects, could take more time to implement compared to the procedural design approach from PPA3.

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.

    - Yes, the code provides sufficient comments or is self-explanatory. The code is relatively easy to understand given knowledge of how the program is supposed to function. The main shortcoming is a piece of code that was left commented out instead of removed before the final submission. Also, more attention could be given towards grammar and spelling.



```
28  objPosArrayList* Player::getPlayerPos() //position getter
29  {
30      // return the reference to the playerPos arrray list
31      // returnPos.x = playerPos.x;
32      // returnPos.y = playerPos.y;
33      // returnPos.symbol = playerPos.symbol;
34
35      return playerPosList;
36  }
```

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.

    - Overall, the code follows a good structure with good indentation and readability. The only thing I would change is reducing the amount of unnecessary white space in the code to shorten the file length and reduce the amount of scrolling.

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)

    - Yes, the snake game has a pretty smooth, bug-free experience. However, a more systematic issue with the game is that each time the board refreshes it feels laggy. A different delay constant or system for the overall game processing could solve this issue.

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.

- No, the snake game does not cause memory leakage as all heap members were released in their respective deconstructor functions.

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 experience overall was smooth. I feel like it definitely helped that we already knew each other before partnering up for this project as we understood each others work habits. There was also an issue where I accidentally made changes to one of the main files that caused an error when my partner tried to push his completed iteration.
    - The experience was pretty good. We delegated tasks and made sure to manage our time and as a result we were not rushing to finish at the last minute. As my partner mentioned, we learned that it is better to not work on one file at the same time as there will be some code overlap issues when trying to update the repository.