COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Elizabeth Yorke	Saad Siddiqi
Team Members Evaluated	Ryan Phiby Mathew	Ishaan Malhotra

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 header files are very clear in portraying how they will behave and what their primary purpose is. The structure of the header files also spaces and groups the members clearly in an intuitive way that makes it easy to interpret what each one will do. One suggestion is to keep the order of private vs public members consistent throughout. I noticed that in *GameMechs.h* the private members come first, then public, while in *Player.h* the public members are first. This can cause confusion for others tasked with updating or maintaining the code, meaning the chances of private and public members getting mixed up is higher.

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.

The organization (i.e. spaces, and groupings) is well done and the code has great readability. Variable and object definitions are clear and grouped with other objects / references they are connected to. The program is written in an intuitive, easy to follow, and procedural manner. Comments were used sparingly and to the point. I do think that while *line 57*, where there are references to two functions in other classes, may be more concise in terms total line count, I do not think it's the best way to call the functions as it can be confusing. Pointers and referencing are a generally complicated topic so I feel like it would be more efficient for updates and maintenance to have this line split in two. I also noticed the **GetInput()** function in the main file calls both **myGM->getInput()** and **myGM->setInput()** but when comparing all three functions, the way their **GameMechs getInput()** is set up makes the if statement in the main files **GetInput()** redundant. Their **GameMechs getInput()** also already set the input by calling **MacUILib_getChar()**, making the **myGM->setInput()** line also redundant. For this section, simply calling **myGM->getInput()** as the only line in the main **GetInput()** function would have been enough.

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:	**CONS:**
- Better for organization - Easier maintenance - Faster and more efficient debugging - Better readability - Easier for iteration updates and changes promoting iterative engineering	- Keeping track of multiple files - Ensuring files are correctly linked where necessary - Adapting to OOD rather than the procedural method used prior - Small syntax differences that require lots of corrections while adapting to the very similar yet slightly different language

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 exhibits a reasonable level of commenting and self-documenting coding style making it relatively understandable. The class and function names are generally descriptive, aiding in comprehending their purposes. For instance, in *GameMechs.cpp*, functions like **getExitFlagStatus**, **setExitTrue**, and **setLoseFlag** are self-explanatory. Similarly, *Player.cpp* includes functions like **updatePlayerDir** and **movePlayer**, which are indicative of their roles in managing player input and movement. However, there are areas where additional comments could enhance understanding, particularly in complex logic. For example, in the **RunLogic** function of *Project.cpp*, the logic involving player movement and collision detection is present but might benefit from inline comments explaining the specific conditions being checked. Additionally, some complex algorithms, such as the **movePlayer** function in *Player.cpp*, could be further explained to help grasp the reasoning behind the implementation.

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.

The code generally follows good indentation practices, utilizes sensible white spaces, and employs newline formatting for better readability. The indentation is consistent throughout the program enhancing the visual structure and making it easier to identify code blocks. Sensible white spaces are used to separate keywords and operators. Newline formatting is appropriately utilized to distinguish between logical sections of the code, such as separating function implementations, loops, and conditional statements. However, there are few areas where further improvements could be made to enhance readability. For instance, in the **DrawScreen** and **RunLogic** functions within *Project.cpp*, the nested loops and conditional statements could benefit from additional spacing comments to clearly outline the different sections of code and enhance understanding. Another example is in lines 128 and 181 where there are multiple conditions. Having multiple conditions can become confusing especially when math is involved. Incorporating brackets to differentiate the cases would have improved the programs overall readability.

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 provided a smooth and bug-free playing experience. The fact that the game functions well, with proper handling of features such as wrapping around the border, detecting food collision, checking for self-collision, and incrementing the score, is a positive indication of a well implemented program. The correct growth of the snake's size and the random generation of food items further contribute to a seamless gaming experience.

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.

Using Dr. Memory, the memory profiling report indicates that there are no memory leaks in the program. The code successfully deallocates all the memory it allocates, and there are no unreleased memory blocks at the end of the program execution. This suggests that the deallocation mechanism implemented in *Player.cpp* and *objPosArrayList.cpp* as well as in *Project.cpp* are effective in releasing the allocated memory, preventing any memory leaks using 'delete' or 'delete[]'.

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.

Collaborating on code for the first time was a bit of a shock. Having never done anything like this before, many *'how'* questions swarmed the air. Being able to push and pull the updates made by each other was so convenient and made sure each partner was caught up on what was edited and how it's effected the code. In other words, it broke the updates into smaller and more manageable parts. It also gives students a great taste of how coding will feel in industry when it's mostly group collaboration. It encourages students to lean into other's design approaches and ideas and practice their critical thinking skills to come up with standards and compromises.

The biggest challenge was in the beginning not knowing how GitHub worked for collaborative and iterative programing. Having a demonstration beforehand and even a small hands-on activity to get accustomed would have been very beneficial.