**COMPENG 2SH4 Project – Peer Evaluation**

Your Team Members \_\_\_Justin Hanna\_\_\_\_\_ \_\_TJ Langlois\_\_\_\_\_\_\_\_\_

Team Members Evaluated \_\_\_\_Tyler Smith\_\_\_\_\_ \_\_Mathew El Chalouhi\_

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 code is very organized, and we can infer inter-object behaviour from the input variables passed in. For example, we can infer that the Player class will interact with an instance of the GameMechs class since a pointer is passed into the function’s constructor. Similarly, a Player will most likely also interact with an instance of the Food class for the same reason. This is intuitive, and memory efficient, since only one instance of each class is required and then is passed around by reference. Good job!

One aspect of this that we found odd was that the Food was being handled in the Player class as opposed to the GameMechs class. We view food as a game mechanic rather than a player mechanic since it’s dependent on the board primarily, so the location of this implementation struck us as a little odd.

1. **[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.

Throughout the code, its pretty easy to see how the objects interact with each other, but in one case it is a little obscure. In Initialize(), the food class is initialized and GameMechs pointer “gm” is passed through it unnecessarily. The GameMechs pointer is only used to get board dimensions in the Food class. It would be better if those dimensions where passed through only using getBoardSizeX() and getBoardSizeY() as integers, since that wouldn’t expose the entire GameMechs class. Since the only spot this would be required is the main function, it would simplify the operations significantly. Throughout the rest of the main function objects only call their own member functions, meaning the only place information is instantiated is Initialize. It works for this implementation, but objects can have some functions passed by value rather than by reference everywhere especially if they’re primitive datatypes.

1. **[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 main pros of an OOD approach are to improve code readability and reusability. Objects encapsulate data and functions and can run functions on their own data members locally without bleeding private information. OOD allows us to keep track of variables separately, such as a player’s positions in a Player class and the game score in a GameMechs class. These two data pieces are crucial to the design of the game, but instead of keeping it all in one cluttered global space we can organize it into classes. We do lose the simplicity of accessing data globally, as we were required to in C’s functional design, but we gain modularity. Need to use an arrayList to hold the Player positions? Make them out of objPos instances. This would have been hard to implement with only a functional approach, as keeping track of all these nesting variables in structs would require much more complexity.

**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.

There are comments all throughout the code, explaining in plain English what the following syntax does. One thing I’d like to see more of is justification for the logic, especially the heap operations in DrawScreen(). I see that the code does run, and the game is playable without memory leaks, but a lack of comments in the operations there make it confusing to understand what’s going on, and why it’s necessary.

Some of the design decisions made in this code also left us confused. For one, all the global variables are set as pointers and allocated on the Heap, such as the Player and the GameMechs. We understand the dynamic necessity of ArrayLists being allocated on the heap, but all the other variables seem to be just fine if left on the stack. There is also an inconsistent use of dereferencing and accessing (“->” vs “\*().”). It is conceptually identical, but sticking to either the shorthand or the long form improves readability and consistency.

1. **[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 code, indentation is used to make For Loops and Switch Cases easier to read. But the code lacks newline formatting in certain areas. In Project.cpp, else if statements are followed directly by print statements which not only decreases the readability of the code but is bad in case of expanding the code to more than one line. It’s a useful shorthand for one-liners, but it isn’t sustainable. By using brackets and putting the print statements on a newline, it would improve the readability and expandability of the code. Lastly, the code utilizes good use of white spaces, especially in-between commas and after semicolons. This makes it easier to read the info computed in member functions.

**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 offers a bug-free playing experience but isn't extremely smooth. There is a large delay when playing and the screen is flashing. This is due to a large number of MacUILib\_prinf() or std::cout calls in one game loop. This could be avoided by calculating, or rather rending, a large string filled with one frames output throughout the DrawScreen function and call std::cout once at the end.  
One other thing to note, which may not be a bug, is that the End Screen when a player finishes the game there isn’t a different message depending on the end case. It would be useful to add differing messaging to note when a player finishes the game due to self-collision or when a player manually leaves the game with a space key.

1. **[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 code does not cause a memory leak. We used Dr Memory to provide a report which informed us the only memory leak was from msvcrt.dll, which is a Microsoft Visual Studio issue. There were also two uninitialized reads that are result of using MacUILib\_getChar() and MacUILib\_hasChar(). These reads access a part of memory that is not initialized yet, but that don’t cause a SEGFAULT error. Once again, they originate from a library outside the code, so the game itself doesn’t cause these issues.

**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.