**COMPENG 2SH4 Project – Peer Evaluation**

Your Team Members Kareem Odeh and Ahmad Molhim

Team Members Evaluated Abdulrahman Butt and Amna Khan

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.

* It is quite easy to grasp the actions of each object involved in the program by glancing at the header files. Positives include its clarity and conciseness. The naming guidelines are uniform. Furthermore, each function name is really brief. getFoodPos is a good example of a well named naming convention as it does exactly what says it does. Without reading the implementation files, it is more difficult to comprehend the intended use and interactions of methods and classes due to the absence of comments and documentation in the header files.

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.

* The main game loop in main() follows a straightforward and traditional game loop structure. This structure is easy to understand and is a standard in game development. A few positives include that the player and food pointers, as well as the game and food pointers, have all been clearly initialized. There is a logical and obvious interaction between the objects. The Player object communicates with Food to monitor food intake and with GameMechs to control the game's state and player interaction. Food is in charge of overseeing actions connected to food, whereas GameMechs manages the overall status of the game. The code lacks error handling, especially for operations that could fail like new allocations. I also would’ve used interfaces to improve the flexibility of the project.

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.

Project Pros: In the main loop, objects such as GameMechs, the player, and food interact through well-defined interfaces. Extremely little global variable. Class organisation of game functionality allows for a clear division of responsibilities.

Project Cons: Even though C++ OOD is much more beneficial in real world scenarios, it is a much harder programming language to master, and is also more difficult to test.

PPA3 Pros: The fact that everything is on a single page eliminates the need for you to search through numerous files.

PPA3 Cons: Even though you might think it is better to have everything on one page, it is also a much less efficient way to encounter code in the real world as we are going to be writing much more complex code with a lot more lines which requires neatness and efficiency.

**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 snippets that have been provided though they are very understandable for an experienced coder, they do need to require notable explanations for the unfamiliar people since this makes it harder to interpret the code functionality. Understanding is aided by the coding style's somewhat consistent naming conventions and structure. This uniformity, though, doesn't entirely make up for the absence of remarks. Understanding can be greatly enhanced by the use of inline comments, particularly in complex functions. For Player::movePlayer, for instance, it would be helpful to describe the reasoning behind boundary checks and the events that take place in each scenario of the switch statement. Including comments in the header files to explain the purpose of each class and its primary responsibilities would also help a ton.

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.

* The code generally follows consistent indentations as well as white spaces which really did aid in distinguishing different parts of the code and make it more readable. The use of new lines to separate logical blocks of code and functions is generally good as well. Use newline formatting more frequently to divide up logical chunks inside functions. Although the utilisation already in place is sufficient, there are some locations where more separation could improve readability even more. To make Player::movePlayer easier to understand, the loop that tests for player collision might be separated from the rest of the code. Think about dividing more complex functions into smaller, more targeted ones. The various situations in the switch statement, for instance, in Player::movePlayer, might be divided out into distinct 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)

* As a whole, there are no significant noticeable flaws in the code while the program is run.

However, one thing I noticed while running the team’s program was the repetitive flickering and glitching of the game board which is not always due to the main program the team has written but if I had to assume the problem actually was in the code, I would presume it to be in the drawing or clearing of the screen. In this specific case I would set in place double buffering implementation where you draw to an off-screen buffer and then copy it to the screen in a single operation. Also, another insignificant flaw I noticed is that the Snake Game isn't responding as quickly as it should be, there are a few possible reasons why, as well as some basic solutions. I would implement a fixed time step in the game loop, this would ensure that the game runs its logic and updates at a constant and consistent rate.

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.

* There is no memory leakage as a result of the team's snake game. All memory allocations have been correctly released to allow for proper memory handling in their code.

A screen shot of a computer

Description automatically generated

**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.