COMPENG 2SH4 PROJECT – PEER EVALUATION

Your Team Members:

Anushka Goswami (goswaa4), Owen Martin (martio6)

Evaluated Team Members:

Vivian Nguyen (nguyev71), Xinyi Wang (wangx634)

PART I: OOD QUALITY

Q1. 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.

A1. The code written by their team is mostly coherent from the OOD perspective.

- Classes are defined well, and objects described by the classes are created, edited, or mutated with adequate use of constructors, setters, getters and mutators.

- There are only some redundancies within their code, for example, the creation of functions checkFoodConsumption and increasePlayerLength, in the Player class, that were simply never completed in the corresponding cpp file (left empty), and also never used/called in the main project execution file.

- A similar issue arises in GameMechs.cpp and GameMechs.h, which contain parameters that never get used such as loseFlag, and getter and setter methods for the same. These parameters and functions never get used in the main project code.

Q2. 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.

A2. Main program code is mostly solid and consistent, with the process easy to follow and understand.

- Class objects have been used correctly to implement different actions based on the players input and the food spawn location. The way different class objects interact is also understandable and logically sound.

- Code works fine but is primitive in nature. One food item spawns at a time, and snake grows larger on consumption, but the snake doesn't check for self-collision. Essentially, the game has no lose condition, which is a little underwhelming.

- Certain areas of the code introduce certain inefficiencies from a game mechanics perspective, for example, every time the executable is run, the food initially spawns at the exact same location (due to the initial position being fixed at initialization). This can be improved by using a time seeded rand() function to generate int coordinates for initializing initial food object position.

Q3. 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.

A3. From an organizational point of view, OOD programming helps make the code modular and allows for greater expandability.

- Addition of newer features and/or existing objects is facilitated within object-oriented code. This form of coding also provides safeguards against downstream interference, meaning that certain features within the program can be made inaccessible/immutable through future edits.

- While OOD based code development has many merits, the development of classes and objects controlled therein is a rigorous and often time-consuming process, especially when compared to the straightforward approach scene in procedural programming.

- Getting multiple class objects to work together cohesively often requires a stringent and well-defined class hierarchy, which needs to be though out carefully and often extends the development process.

PART 2: CODE QUALITY

Q1. 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.

A1. The code offers sufficient comments to allow observers to parse the function and logical flow of actions within the main program loop.

- The use of self-documenting naming conventions for variables, functions and other objects also facilitates the observer to understand the process flow.

- There are certain areas within the code, where trial code (from development process) has been left inside comments, for example, within the DrawScreen function in Project.cpp, where a block

of code concerning a 'Lose' message has been commented out. It is not possible to say whether that was intentional or not.

- Some other areas in the code also have certain snippets of code commented out, which appear to be remnants from trial testing for some cases but lack any description at other points, making their intent ambiguous.

Q2. 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.

A2. The code has been organized to be legible and understandable with conventional use of newline characters, indentations and whitespace characters.

- Everything has been mostly adequately commented. Apart from comments left over from the developmental process (lines of code in comments), the code has been well annotated. Most functions, variables and blocks of code contain a comment regarding use and function.
- It might be beneficial for their team to remove trial-testing code, and debugging comments from the program before submission, as they serve no purpose and do not add to the logical flow of a final product.
- Some classes such as Player.cpp contain unnecessary functions that never get used and never get initialized (empty inside). Such functions and/or snippets of code ought to be removed.

PART3: QUICK FUNCTIONAL EVALUATION

Q1. 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.

A1. The snake game offers a mostly smooth, bug-free playing experience.

- One minor inefficiency visible in the final game executable is that the food generation gets slower and slower the longer the snake grows. This is because the food generation occurs after a collision check integrated within the move player function in Player.cpp.
- The way the code is structured causes the food to generate with a delay after the snake head completes a collision with the previous food item.
- To fix the lag, it might be a good idea to re-evaluate the logic process of how the generate food function is called upon detection of a collision, and how the food is printed withing DrawScreen.
- To get a good idea of what causes a delay in change for the food spawn position, it might be a good idea to go over the collision logic code, the generate food code, and the DrawScreen code within the main program code.
-

Another minor improvement that may be made to the game.

- Game has no self-collision mechanic.
- It might be a good idea to implement a self-collision mechanic by creating a function within Player.cpp to check for self-collision instances.
- This would also allow their team to use their loseFlag parameter within GameMechs.cpp, which is instantiated but never used, as the program currently does not contain any lose condition.

Q2. 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.

A2. No, their program does not cause any memory leakage. We tested their code using Dr Memory, memory profiler, and did not find any warning caused by memory leakage. There were 0 bytes of memory leak detected.

PART4: YOUR OWN COLLABORATIVE EXPERIENCE

Q1. 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.

A1. This project was a good introduction for the way collaborative coding works. We were previously unexperienced on working on a large-scale coding project collaboratively on this scale. It was notably different from solo coding projects, where one programmer gets to formulate the entire code structure themselves and make all the decisions for implementation.

Different programmers have different coding styles, the way one team member might choose to implement a feature may be completely different from the way their colleague chooses to do theirs. This can cause friction during code integration processes.

Having programmers versed in different sets of programming techniques works for the benefit of the code in the long term. Discussion is often a crucial component of finding even ground, and while this may cause trouble initially, it often leads to a comparative discussion where programmers can brainstorm to find the best method that works for all parties involved, creating code that may be better than what either might have chosen to create.

Since both of us were communicating our progress and keeping track of each others code, it was fairly easy to reach productive solutions. All in all, it was a fun project, and we were both fairly pleased with the outcome.