COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members: deyn1 mazumm4

Team Members Evaluated: yuehj dosanj5

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **25 marks**. Do not exceed 2 paragraphs per question.

Peer Code Review: OOD Quality

1. **[3 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 program loop can be found in Project.cpp. In this file, the coders have properly used objects "myPlayer", "myGM", and "myFood". The "myPlayer" depends on the "myGM" to control the boundaries and speed, whilst "myFood" is used to interact with "myPlayer" when it comes to generating food and collisions with the boundaries. All these objects have been stated clearly amking the code easier to read and understand. Another positive aspect of their code is that they used pointures to ensure that the objects stated above are created on the heap. In addiiton, they have used the object "myGM" such that it can track the players score, speed and if the player wants to exit the game. This helps to make a better and friendlier interaction between the gamer and output of the game. One negative with the code in the main loop is that the use of new and delete pointers can cause memory leaks in the code.

2. **[3 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.

C++ OOD makes it much easier for collaborative coding as well as its ease of using classes and objects throughout the code. Not only that the classes hide the implementation details making it easy for others to read the code in comparison to the C procedural design in PPA3.

However, one of the cons of this is that it requires more memory usage in the objects.

Peer Code Review: Code Quality

1. **[3 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.

Both programmers did an excellent job when it came to commenting throughout their code. As a result of such sufficient comments, it was easy to understand how the objects interacted with each other to make the game work. The comments also allowed for easy access to find where and how iterations, incrementation, and the overall logic used to make the snake game work. This included how the snake would grow, how the food would appear and so on with the use of objects.

2. **[3 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 demonstrates good formatting overall, with consistent indentation and logical use of white spaces. While minor inconsistencies exist, they do not detract significantly from readability. For instance, some functions, such as Food::generateFood(), lack uniform spacing between logical sections, which can make the structure harder to follow. Additionally, comments are sometimes cramped against code, as seen in // Generate new food position without a preceding newline for separation. Moreover, brace placement in methods within GameMechs occasionally varies, with some being on the same line as the function header and others on the next. Improving uniformity in formatting across all files, such as consistent brace placement and spacing, could further enhance clarity and make the code easier to navigate.

Peer Code Review: Quick Functional Evaluation

1. **[3 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 technical user feedback)

The Snake Game offers a generally smooth playing experience. A possible bug might be related to food generation, where food may spawn on blocked positions or not regenerate correctly after consumption. These issues could stem from improper boundary checks, inconsistent update rates, or flaws in collision detection logic. To resolve them, reviewing the movement logic, ensuring proper boundary conditions, and debugging food generation code with additional checks would likely help improve game stability.

2. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.

No, the Snake Game does not cause memory leaks.

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

No it is not sensible because there is no need for a struct since there were only three classes.

2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. You are expected to facilitate the discussion with UML diagram(s).

I would change the objectPos by replacing the struct with two integer variables a and b.

objPos
- a:int - B: int - Symbol: char
+ objPos() +objPos(aPos:int, bPos:int, sym:char) +setObjPos(o:objPos):void +SetObjPos(aPos:int,yPos:int,sym:char):void +getObjPos():objPos const +getSymbol():char const +IsPosEqual(refPos:const objPos*):bool const +getSymbolifPosEqual(refPos:const objPos*):char const