COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members _James Shakespeare_ __Taejin Eom__

Team Members Evaluated ___David Elyahky____ _Vince Simone Amado___

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.

   One observation about the object interactions is that the program will often call a class method as a parameter, instead of assigning it to a variable and then passing the variable. This has some positive and negative effects. One positive effect is that this avoids using excess memory when storing values to variables. One issue this causes is that it is more difficult to read and understand what is being passed into each function. Another issue is that it can waste processing power if a method has to be repeatedly called to get the same output, instead of just saving the output.

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.
   a. **Pros:**
      i. Easier to modify or scale up. This is seen when going from position to position-array-list.
      ii. Private members are harder to accidentally access.
      iii. Easier memory de-allocation.
   b. **Cons:**
      i. There are more rules and syntax to learn. One example is the rule of 4 minimum/ 6.
      ii. C++ OOD can be slower.
      iii. Can lead to overcomplicating things when not careful. An example of this is how a position type is made instead of just including it as a field.

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.

   The code has comments in the food and player class files, which improve readability by explaining what certain for loops, variable, and functions do. If these comments were also

implemented throughout the rest of the project, the code would become more readable everywhere. In classes like GameMechs, this isn't as necessary, as they only used getters and setters, which are self explanatory. The objPosArrayList class could benefit from more comments especially on the more complex methods like insertHead, insertTail, removeHead, and removeTail.

Another way this group made their code more readable was by using self explanatory variable, method, and data member names. Methods like checkSelfCollision, increasePlayerLength, and isPosInList clearly state their purpose. There was one exception to this in the increasePlayerLength function where an object is simply named 'object' with no explanation. While detailed and specific method names are easier to understand, they can make the code appear complicated. One way to avoid this is by doing method overloading. For example, the methods to generate regular and special food could have both had the same, simple name, "generateFood" if the programmer had a parameter that is only inputted when special food is being generated.

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.

   Yes, the program follows good indentation, white space, and newline formatting. All of these contribute to the readability and flow of the code. They make it clear which parts of the code relate and interact with each other. Indentation is used consistently throughout the project and is rarely forgotten. White space is forgotten occasionally. Some cases of this are where a variable is written directly next to an equals sign, with no white space. However, this doesn't cause any issues with readability and is only a very small issue with continuity. There are more cases where newline formatting is not done between different segments of code. One example of this is in the increasePlayerLength.

   These white space formatting issues can be fixed by simply focussing on consistency. While this is difficult in a group project where different programmers write with their different styles, it is possible to review code near the end to ensure everything uses the same formatting conventions. The newline formatting can be fixed in the same way. This method would mean that the programmers would notice that the variable instantiation done in the inreasePlayerLength method was not separated through newlines, unlike in other methods.

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)

By running through the code multiple times, there were very tiny bugs related to the bonus part of the assignment. In the rubric, it states to generate at least 3 food for the snake at a time. However, in the game, it would occasionally spawn 1, 2 or even 4 food at a time. Most likely, there is some sort of error in the generating food logic in the food class. Furthermore, it could be something that has to do with checking the food position with other pre-existing food or even the player. There could be an error in logic where sometimes if they overlap, the program doesn't generate a new position for the food and ends up not printing over the food that was already there. For cases where 4 food spawn at a time, it could be that the existence of special food and food aren't checked correctly and sometimes prints all 3 of the food plus the special food. This isn't particularly wrong, but will make the game more inconsistent.

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.

   After running drmemory on the code to check for any memory leaks, the results concluded that there were 0 bytes of memory leaks. This means they properly deleted all the memories they made on the heap.

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?

   The compound object design of objPos class is not sensible for a project of this size. This is because the main benefit of the compound design is that allows for the pos struct to be used outside of the objPos class. This is not required to be done for this program where only objPos class objects have a position field.

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. <u>You are expected to facilitate the discussion with UML diagram(s).</u>

   It would be better to include the x and y positions as fields within the ObjPos class. This means memory allocation becomes less complicated, referencing position becomes less complicated, and the code becomes clearer. An example UML diagram can be seen below.

ObjPos
+ xPos: **int** + yPos: **int** + symbol: **char**

+ objPos()
+ objPos(xPos:**int**, yPos:**int**, sym:**char**)

+ setObjPos(o:objPos): **void**
+ setObjPos(xPos:**int**, yPos:**int**, sym:**char**): **void**
+ getObjPos(): **objPos const**
+ getSymbol(): **char const**
+ isPosEqual(refPos:**const** objPos*): **bool const**
+ getSymbolIfPosEqual(refPos:**const** objPos*): **char const**