COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members __sarkeb1______ __hasanz15____________

Git Team Name – 404 Partner Not Found

Team Members Evaluated _duam4_______ _mustam26_____________

Git Team Name – Hackstreet Boys

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.

   Yes, the code is easily interpretable when gone through, regardless of the comments added into the code. One positive aspect of the main program loop is that it doesn't include a lot of the code, only the aspects that are absolutely necessary. The groundwork of things such as the player objects/array list and the food objects are left to their respective pages and are not integrated into this main page, which removes clutter and therefore leaves the main page to be interpreted properly. Even though this may be a nitpick, since this team's implementation of the DrawScreen() was a bit different then ours (which may be the case for other users as well), perhaps a little more detail would be helpful, since this team proceeds to create the snake and the food objects before the game board, which might not make sense intuitively, even though it doesn't make a difference in the implementation.

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.

   One of the biggest strengths of OOD programming is the organization involved with the code. If you are looking to create a big program which requires a lot of code and different moving parts, you can put your code for these different sections in different pages, so if you have problems concerning the implementation of any one of these attributes, you have an immediate understanding of what is wrong with the program. Therefore, OOD in most instances makes coding organization and troubleshooting/debugging easier. It also makes it easier to track the relationships between different sections of code, since you don't need to look through a massive amount of code in order to understand how the different sections interact with each other. However, the cons arise when OOD is taken too far and objects are used inside more objects, and your project becomes an interconnected web of relationships, which may actually become harder to understand than an entire page of procedural programming. This may also make troubleshooting very hard, especially in the final page/product where all the relationships come together, since problems rooted in the deeper levels of code might be harder to figure out. This make debugging a more intensive process in OOD since you first have to test out the part themselves, then also test out the relationships between those parts.

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.

   Yes, the code offers sufficient comments to help us understand the code functionality more efficiently. One instance of this is in objPosArrayList.cpp, where every void return function has an explanation in order to help the user/viewer understand the purpose of the function. Another instance of this proper comment usage can be seen in the checkFoodConsumption() function in Player.cpp, where both variables and blocks of code are commented in order to understand their purpose and functionality. One final example of proper comment usage in the code is shown in Project.cpp, where comments are incrementally every 2-3 lines in order to make the thinking process/functionality of the code more intuitive to the user.

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 code follows good coding conventions and principles. Examples of this can be observed throughout the code. One example of this is in the main Project.cpp page, where whitespaces are provided between main sections of code, and newline formatting is used consistently throughout the page. Perhaps the DrawScreen() block could have better whitespace implementation, but that is a minor concern. We would put in more whitespaces to separate out the different sections of code representing different objects (snake, food, gameboard). Another example of proper readability is the objPosArrayList.cpp file, where all 3 of these proper coding practices are observed.

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)

Yes, the Snake Game offers a smooth, bug-free playing experience. The snake object itself moves around smoothly and responds to keyboard inputs well. The snake also grows in length appropriately as it eats different kinds of food, and the movement of the snake does not get affected. As soon as the snake runs into itself the game over seen shows with a custom message, and the game over button with the space also works.

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 have any memory leaks.

<u>Project Reflection</u>

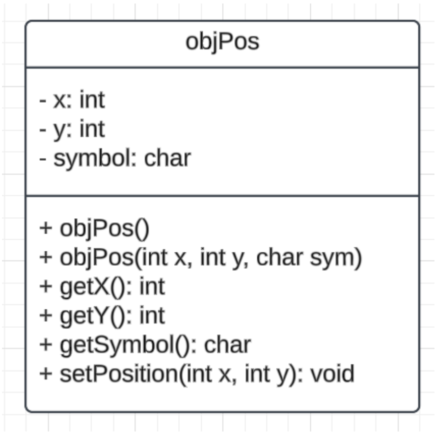
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

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

The objPos class is not sensible because it is overly complex since the use of Pos struct results in dynamic memory management which is unnecessary to store x and y positions. Needing to allocate and deallocate memory for the Pos struct can cause inefficiencies and higher risk for memory leaks especially for any frequent operations.

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>

We can improve the objPos class design by removing the struct and instead just using int variables to store x and y positions as shown in the revised UML diagram below.


objPos
- x: int
- y: int
- symbol: char
+ objPos()
+ objPos(int x, int y, char sym)
+ getX(): int
+ getY(): int
+ getSymbol(): char
+ setPosition(int x, int y): void