COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Andrej and Amhar

Team Members Evaluated Justin and Eric

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.
   -The naming of different objects is relatively well done throughout the project which follows the rules of object oriented programming since they can easily be interpreted. For example objects such as "CurrHead" or "CurrFood" clearly indicate that the object is used for tracking the current position coordinates and the symbol of both the snake head and the generated food. Another example is a variable named "GetNumFood' which is used in a for loop as an upper boundary for detecting the number of generated foods that should be printed on the screen. The variable is for exactly what its name represents. Some negatives that we noticed are in the updatePlayerDir() function, the cases only take into consideration lowercase letters w, a, s, d, if the keyboard input is in the uppercase letters, the cases wouldn't be functional. We are also not sure how beneficial it is to combine the game mechanics and food classes into one bigger one, it makes the code less organized and harder to manage.

2. **[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.
   -Yes, it can be quite easily interpreted how the logic works. First the initialize function is run which is a good thing since it first of all clears the screen from any previous printouts, it initializes all the components such as libraries and pointers so that they can be used throughout the compilation. Then a while loop is started which is beneficial in this case since the game has to be running the whole time, until a losing condition such as the snake running into its own body is met, which terminates the while loop and goes into the cleanup function. Inside the while loop, the first function gets the input from the user, then runlogic(), which updates the snake's direction and moves it according to the key pressed, then the drawscreen() function is run so that the following updates can be shown visually on the screen, and in the end there's a 0.1s delay before running through the while loop again which basically represents the frame rate at which everything happens. The naming of the functions in the main loop makes it very easy to interpret what each function does and the order it runs in. I believe they implemented their own code very well into the run logic, all the functions were put into their right place, if they contain a fair amount of lines, they made sure to add

comments so that they can be easily interpreted.

3. **[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.

Object Oriented

Pros	Cons
-Code can be reused by using inheritance allowing functionalities from classes to be used multiple times throughout the code -Encapsulation makes the code much better organized, and easier to read and understand	-Not as easy to learn in the beginning as procedural - Large-scaled projects can be hard to manage if not properly designed -Same variables might have to be repeated

Procedural

Pros	Cons
-Procedural is much more simple and easier to understand if you are a beginner -Easier to track when debugging since it's executed in order mostly -Easier to understand and modify when doing smaller projects that don't require too many lines of code	-Much less reusability freedom compared to object oriented -As the code grows in the amount of lines, it becomes messy and hard to understand

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 is mostly commented to a great extent. It explains what every single initialized variable and object is used for, it describes which variables are on the heap, it explains what each if-else condition is used for and the reason it was used in that part of the code, it describes checking conditions such as checking that the list is not empty before removing something from it so that compilation errors are avoided, in more complex functions such as generateFood(), every for loop is thoroughly described for what it does, and again if statements are very well explained for what their function in the code is. If there is any shortcoming we observed, it would be not adding enough comments in the main function. Even though we as students by doing PPA's learned how the main logic works and what each function such as Initialize(),DrawScreen(),Cleanup() does, someone who never took this course

and wasn't quite familiar with the code might misinterpret their meaning, which is why it would be a good idea to also add comments next to them and write down what exactly is being initialized, how the function exactly draws the screen and which tasks are done in the cleanup procedure. Another good thing to do would maybe be to define integers such as the total number of foods as constants so that they can be used across the code multiple times and avoid error when writing them over and over repetitively.

2. **[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.
   -In most of the code, it follows good indentation making it easy to follow the structure and logic. White spaces are used effectively, enhancing the readability of the code. The use of new lines also helps make it easier to read. However, in the "GameMechs" file, specifically the GenerateFood() function, the extensive code could have benefited from additional new lines and the grouping of code blocks. This would have increased the readability of code significantly. The code is well commented throughout, providing clear explanations of each part's logic. Overall, the code follows good practices across the entire project.

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)
   -After running the game I've found it to be very smooth and no bugs were visible. I really liked how they implemented the game when it comes to food generation, at the beginning the amount of food generated on the screen is not too high, and then suddenly after you pass a certain score there is a lot of food generated for a short amount of time, this makes it harder for you to see where the snake is, which I find to be a positive thing since it makes the game more challenging and fun. The only issue I have as mentioned above is that in the input cases only take into an account the lower cases of letters w,a,s,d, and while playing the game if I suddenly click the CAPS-LOCK button and letters are changed to uppercase, the input is no longer registered and I'm unable to move the snake. The second thing I found that's not necessarily an important issue is that the final score is displayed for a very short amount of time and then it disappears. I believe it would be better if it was displayed for at least 3-4 seconds before clearing the screen so that the player has more time to see it. Other than that, all the features such as self-collision detection, score tracking, wraparound, collision with food, etc. work exceptionally well.

2. **[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.
   - **Fortunately, the Snake Game does not have any memory leaks. This is due to the correct implementation of destructors for both the "Player" class and the "objPosArrayList" class.**

Furthermore, a destructor was not needed for the “GameMechs” class as there were no members declared on the heap. The main program, specifically within the “CleanUp()” function, the heap members declared in the “Initialize()” function, were appropriately deleted from the heap.

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.
   -We were working as a team, and one of the things that worked very well during our collaboration is that we were able to work on different things with the help of object-oriented programming without interfering with each other's work. For example, I(Andrej) worked on the object position array list while Amhar worked on the random food generation, and as long as both of our codes passed all the necessary tests and worked properly, we were able to later combine them in the main function without making any big modifications. This would be very hard to do in procedural programming. Even though we both worked on different sections, we still made sure to explain to each other what each section of the code does so that we both understand everything thoroughly. Working as a team was also beneficial because whenever one of us got stuck on some part of the code and couldn't figure it out, the other person would help, having ideas from 2 brains is always better than from just 1. If I had to give a negative aspect of working with a teammate, it would be that it brings a lot of stress since your teammate is depending on you, and you have a lot of pressure to finish everything quickly since they can't move on until you finish your part of the code, but this is also a positive thing since it helps us get used to working under pressure and stress, which simulates how it would later be at our job in the real world. Overall, it was a very good experience and we worked excellent as a team.