COMPENG 2SH4 Project – Peer Evaluation

Your Team Members: Erion Keka and Robert Rakanovic

Team Members Evaluated: Aurora Boone and Maeve Wheaton

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

git clone https://github.com/COMPENG-2SH4-2023/2sh4-project-maeve-and-aurora.git

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 header files are easily interpretable. The functions that are declared within the header files follow good naming practices. From the name of each function in the header files, you get a sense of what the function entails and are placed alongside other functions that play a role together in the program. This shows great organization by the group which can help the reader understand what is going on with the code without looking further than the header files. For instance, in the 'GameMechs.h' file, the function calls such 'bool getExitFlagStatus()' and 'void setExitTrue()' are placed in an organized manner allowing the reader to understand their roles in the program. Furthermore, as new functions were created and declared in the header files, the group added comments to explain the purpose of the function. This is a great method of providing insight to the reader of the code on the roles of the new functions. The comments placed alongside the new functions are not excessive and summarize the purpose of the function in a concise manner. For example, in the 'GameMechs.h' file, the group created and declared the 'int getGameMechsD(int i, int j)' function. Without a comment, they may be deemed difficult to interpret its purpose within the program; however, the added comment of "Get the ASCII integer stored at a specific address", perfectly explains the purpose of the function. A critique for the group may be to add comments when referencing another class within the header file. For example, the group made a reference to 'objPosArrayList', but did not provide a comment alongside its reference. This can potentially pose a hard time for the reader if they have not read the other files of the program yet; however, overall, the program features sensible code modularization.

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.

In the programs main logic, it is easy to interpret how the objects interact with each other. For example, in the GetInput() function, it is easy to understand that when the GameMechs object grabs an input, if the input isn't a spacebar press, the player object attempts to update the player direction. In most function calls, comments are explaining what the called function does, which makes it easy to understand what each object is doing and how it affects the behavior in the next object but this does not happen for every function. For instance, in DrawScreen() the food and player positions are both grabbed and their locations are checked as explained in the comments, but later in the line 'GameMechsPtr -> reset();' there is no explanation for what this is doing and why it is being used right after grabbed the food and player position. Adding additional comments in places such as these would allow the program to be easier to understand.

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.

In C++, the OOD approach is very useful. Firstly, classes can have access specifiers, such as private, public and protected. In a class in C, everything is public without no option for public or protected class members. Protected and Private class members in C++ are useful for not allowing modifications to an object from external code, but only from accessing it within the class (or derived class as well if using protected). Additionally C++ has something called inheritance, where a class can "inherit" all the properties of another class. This is useful for creating a class with similar functionality, and reuses code efficiently. C does not have this option. C also has very simple memory allocation, using new and free, which is much simpler than malloc calls! However, some cons could be that OOD in C++ can be quite complex, as there can be multiple classes and inherited classes, each with their own private, public and protected members which is hard to all keep track of.

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.

In the project.cpp file, it isn't difficult to understand what's going on with the comments provided, and parts of the code that don't have comments are pretty easy to understand by just reading the functions (For example, the GetInput() function's else statement which has playerPtr->updatePlayerDir(), clearly updating the players direction if spacebar isn't pressed, and the GenerateNewFood() function being simple since it just loops a generate food command 5 times). There are some comments which are a little unnecessary, such as the ones next to the print statements, but these don't really pose any issue. Although most of the files had sufficient commenting, objPosArrayList.cpp and GameMechs.cpp did not have many, which is problematic since the functions they include are very important for the snake game. For the first few GameMechs functions, there are no comments whatsoever. Our group could understand what the functions were doing, but only after reading each individual line of code. Comments would be extremely useful here to make the functions easier to understand. The same applies to few of the objPosArrayList functions, many of them don't have any comments, it would be nice to have an explanation for what the function is doing. For the functions such as removeHead, removeTail, insertHead and insertTail, it should be explained why there is a return when sizeList <= 0 or if sizeList == sizeArray. Without comments here it is difficult to understand why these variables are important and why nothing should happen if the condition is met.

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.

The code has very good indentation and use of white spaces, which made it much easier for me to understand what was happening in each part of the code. However I did notice a few odd things, such as the for statement in the GenerateNewFood() function in Project.cpp not being consistent with the other for statements, having an extra space before i=0. Some for functions have spaces such as i = 0 but many others don't and show as i=0, it would be nice for them all to be consistent (preferable with a space, since many of the conditional statements in the for loops are all spaced as well and it is slightly more readable).

Part III: Quick Functional Evaluation

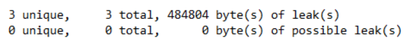
1. **[8 marks]** Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just a technical user feedback)

The Snake Game does offer a smooth, bug-free playing experience. The game runs well; however, there are two critiques that our group has come up with. The first critique that we have for the group is that upon exiting the game, the screen is not cleared meaning that all items remain on the screen after completion. (Either losing or exiting). By removing these items from the screen, the group can provide a more user-friendly experience to the players of the game as the end-game screen would feature a lot less clutter. The second critique that we have for the group is that the group does not feature a lot of comments. Many of the comments placed throughout the code are for the easier functions that may not necessarily require comments for them. A proposed solution would for this would be to add more comments to some of the more complicated lines of code to allow for a better understanding of the code. This would help the partners return to the code and understand what is happening whilst also allowing for other readers of the code to understand what is occurring without much knowledge of it. In conclusion, the code is very well written as it provides a great bug-free playing experience for the players; however, there are some nit-picky changes that can be made to improve the code in the future.

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.

Yes. There is a memory leak present in the Snake Game. There is 3 leaks total of 484804 bytes. The first of the three leaks occurs with line 7 -- 'aList = new objPos[sizeArray];' of the 'objPosArrayList::objPosArrayList' function that is present in the 'objPosArrayList.cpp' file. The leak is present in the 'Project.cpp' file where the objPosArrayList is declared in the global scope as 'objPosArrayList playerPositions;'. The second memory leak is related to the same line in the same function; however, the leak occurs during the global declaration of 'objPosArrayList foodPositions;' in the 'Project.cpp' file. The leak present with these two global declarations means that the memory allocated for this initialization is not correctly cleared. The second memory leak is related to the same function. Both the first and second leak present a direct leakage of 2400 bytes each. The third and final memory leak consists of 2404 bytes leaked directly and 477600 bytes leaked indirectly. This leakage occurs with line 12 -- 'playerPosList = new objPosArrayList[200];' in the 'Player::Player(GameMechs* thisGMRef, Food* foodBinRef)' function that is present in the 'Player.cpp' file. The leak occurs in the 'Project.cpp' file where the 'objPosArrayList' is declared in the global scope as 'objPosArrayList foodPositions;'. The memory allocated for this global declaration is not correctly cleared which is causing the leak. In conclustion, all three of the leaks occur due to improper memory management regarding the global instances of the 'objPosArrayList' as they are not being properly deallocated.

Snippet of DrMemory Results:



```
3 unique,     3 total, 484804 byte(s) of leak(s)
0 unique,     0 total,      0 byte(s) of possible leak(s)
```

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.

It was a nice experience working on a collaborative software project. All the steps in the manual helped us divide the work evenly amongst ourselves to make our work as efficient as possible. One issue that arose during the experience however was working collaboratively simultaneously. It was difficult to work together at the same time on the code unless you were either on a call or with the other person. This made it difficult to work on portions of the code for the Snake Game because either partner may have been waiting for a response from the other programmer to finish writing the code. After researching solutions, there are software solutions such as the Live Share extension that can allow partners to work on their code simultaneously from anywhere in the world. In the future, we would love to take advantage of extensions like Live Share to continue to improve our collaborative experience.