COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Ilya Plashnitsa and Alexandros Tourloukis
Team Members Evaluated	Nikha Jinosh and Hashan Rex

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.

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.

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.

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.

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.

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)

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.

Part I: OOD Quality

1) There is nothing wrong with your header files as far as we could tell. All of your methods and variables had intuitive, reasonable names that quickly and easily convey what they do to other programmers.

2) There are no major problems with the flow of the logic. That being said, you didn't consider the case where the user is able to create a very large snake. Programming is all about planning for the contingency of your end user doing anything and everything that would cause your code to stop working. Your board is 30 characters wide and 15 characters tall. That means you have 364 possible slots to generate food on your board (28x13 to account for the border). Technically, your snake SHOULD be able to reach a length of 364, but you capped the size of objPosArrayList at 200. In addition to allowing the snake to reach its maximum possible size (364) you should have some way of gracefully ending the game when this occurs. It is not enough to simply change the size of objPosArrayList. Since there is no empty coordinate for the program to generate a new food once objPosArrayList has 364 filled elements, your program would enter an infinite loop of constantly generating food on top of the snake "body", checking every element in objPosArrayList, then regenerating a new food on another occupied coordinate. You should also implement a way to break out of this infinite loop.

3) There are advantages and disadvantages to using the C++ object-oriented design implementation over the C procedural implementation. C procedural implementation requires less computing power. This is essential when attempting to run a program on a device with minimal computational power, such as an integrated circuit, or an older device. C++ object-oriented design is more organized and is easier to understand. This feature is favoured when collaborating with other programmers on the same project. Additionally, C++ object-oriented design has the advantage of using every C procedural design feature in addition to object-oriented design exclusive features. Essentially, C++ has the advantage of extending almost every single C feature allowing programmers to write programs that are part procedural, and part object-oriented such as this program.

Part II: Code Quality

1) The commenting was almost perfect. You struck a nice balance between being concise and not omitting any crucial details in your code that a fellow programmer would need to understand your program. Our only criticism is that you left some comments from earlier iterations that you really should have removed. For example, in Player.h, you left a lot of comments that said, "Upgrade this in iteration 3**".**

2) Most of the indentation is fine except for some parts. The only part that was indented in an unconventional way was the initial setObjPos in the Player.cpp where it was slightly difficult to understand that it was all a part of the same function. White spaces were sensible and were

added in areas to make sure the reader understands the different group such as initializing the variables and the functions that use the variables to copy something or run something. Readability was a major issue as it was slightly difficult and more annoying to read the code. Several times, the group uses an if statement to return a void function to end it sooner. This way of writing is redundant and a way to fix this is if they write an if statement with an opposite statement. An example could be that rather than checking if the array is full and if true, returning, if false, running the code: check if it is not full, and run the code; if it becomes full, the if will not register it. Writing it that way makes it shorter, easier to read, and understandable from the very beginning of why that statement is in the if statement. There is also slight repetition of the code with function that do the exact same but have different names (setLoseFlag and setLoseTrue are identical).

Part III: Quick Functional Evaluation

1) There are no bugs per se since the code does run without crashing however there are parts of the game that do not work as they should. When you collide into yourself, not only does the code set the exitFlag to true instead of loseFlag, but when loseFlag is set to true, the user does not have enough time to see the message projected before the code wants the user to exit (line 124 Project.cpp). There is a part of the code that is unnecessary and that is the creation of a new objPosArrayList in Player.cpp line 120. It is not only unnecessary as the code could be written by just using the playerPosList that was already created, but since the program is forced into creating something new and then deleting it and doing that over and over again when eating a piece of food, this slows down the code slightly which could have been prevented. In addition to needlessly increasing the computational resources demanded by this program, creating a new objPosArrayList whenever a new food is eaten increases the chances of another programmer causing a memory leak while collaborating on this project.

2) There is a 2-byte memory leak in Project.cpp . You allocated memory for myGM and myPlayer at the start of this file, and you never freed this allocated memory. You were so close to fixing this problem in lines 143 and 144, but for some reason, you commented these lines out. Although drmemory says that the leak is from MacUILib, take those results with a grain of salt. This 2-byte memory leak goes away when we uncommented lines 143 and 144 in Project.cpp. The problem is actually in project.cpp.

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.

Ilya: For the first collaborated software project, I had a great time. I personally have had pretty bad experiences working with partners since I would usually get the "do-nothing" partner or the "actively-hinder" partner and so I would end up doing almost all of it since I understood the topic and submitting it with the other persons name so that it was officially a "group" project. Working with a partner who understood the topic and was actively involved in the project, either telling me info or asking me questions, was really nice to have. What worked was communication and understand where each of us were in the project, making sure we were both aware of how far the project was along and what changes we made independently. What did not work was trying to get the project to the other persons hands using git push or git pull. It worked once when I used git merge but what we sometimes resulted to is copy and pasting the code to each other since it was easier to do so.

Alexandros: The first collaborated software project went really well. I had an amazing partner, and our skills complemented each other quite well. My partner wrote most of the code, and I implemented most of the logic and testing. As someone with bad experiences in the past working at the undergraduate level in groups, it was quite refreshing that this project went so well. The tutorial videos were extremely helpful, as were the TAs at office hours. I only have two minor criticisms to offer. The first is that the teaching team could have been better at communicating what was expected to be done and in which file. It was a little overwhelming at the beginning navigating all those different files considering how much simpler the PPA and lab files were, but this issue was quickly resolved. My second criticism is that our antiviruses prevented myself and my partner from using drmemory in VSCode. Before my partner and I submitted, I was obsessively checking our program for potential memory leaks. I checked using windows PowerShell after the submission deadline, and fortunately, we had no memory leaks. I am not sure if this was announced in class, but this project would have benefitted from making sure that everyone knew that we could run drmemory through windows PowerShell maybe through an announcement on Avenue.