COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Ben Fabella – Justin Feener

Team Members Evaluated Ian Cho – Maggie Xie

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

Part I: OOD Quality

1. **[5 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.

   After examining our group's header files, it is clear what their objects functions were, and how they would work with one other. Given a player, objPos, objPosArrayList, and GameMechs header files, its clear that the game would deal with some sort of controllable player that would be moving, and from the objPosArrayList header, you can maybe observe that they would be moving multiple objects at once. Overall, the header files are very well organized, and just by looking at the naming conventions for certain methods, you could gain a rough idea of what they're trying to do.

2. **[5 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.

   Given their object-oriented implementation in the main project code, and their naming conventions for variables and methods, it is easy to interpret what is going on and how the objects are interacting with each other to produce the final game. For example, the main function is at the very top of the code, so you can get a breakdown of the main game logic and see that different sections like get input and runlogic have their own functions etc. Given the structure of their objects, it is very clear to visualize what lines of code are doing what. For example, an if statement that has if(myPlayer->checkSelfCollision()) == true)... I can see that they are checking to see if the player has collided with themselves, and if they have then execute a certain set of code when that happens. Although everything looks very neat and organized, we think that their code could have been even neater. For example, in the previous line I stated with if(myPlayer->checkSelfCollision() == true), since checkSelfCollision() is already a boolean that returns true or false, they can take out the '== true' part since an if statement itself will only execute if the condition is true. Other than very small things, the main game loop seems to be efficient and organized.

3. **[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.

   The main differences between the two are the following: C++ has objects which allows adding new data and functions very easy. The OOD organizes code much better than C, and on top of that it is much easier to follow and understand code that you yourself are analyzing. For example, our main project code was under 200 lines of code, while PPA3 for the most part had around 500 or so lines of code. Much less global variables were used with the OOD approach since many things that were once global variables were put in individual objects themselves. For example, the exitflag was put under the Game Mechanics class, rather than being a global variable like in PPA3.

Part II: Code Quality

1. **[4 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.

   Their code has comments for every method and explains logic for code that isn't easy to interpret upon reading at first glance. Therefore, there are enough comments that further explain the logic and purpose of certain lines of code. There doesn't seem to be any shortcomings, as their code is overall well commented.

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.

   Majority of code follows good indentation, however something we noticed that isn't necessarily bad for some, but some lines of comments extend far out, where if you were using split screen options, you wouldn't be able to completely read the lines without scrolling left to right. As previously mentioned, that isn't necessarily bad, but if you were to always use split screen and want to read everything just by scrolling up and down, you couldn't achieve this. Indents are used for code within curly brackets, therefore readability is neat and organized.

Part III: Quick Functional Evaluation

1. **[6 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 their game, we have noticed that anytime a food is consumed, the border in the top right corner extends out by 1 which shouldn't happen. Also, when eating the food, it takes a few frames for the snake to grow, and takes longer to add the food when the snake is larger. The goal would to be having the snake grow instantaneously, however somewhere in their code is causing it to take some extra time/frames/loops to work. Apart from assuring that the objPosArrayList is working 100%, we would recheck the checkSelfCollision(), movePlayer() and the process of increasing the players length, as these are the methods that are running when a player does collide with a food. Maybe it's the way or order the players length is increasing that is taking it longer than it should.

2. **[4 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.

   Although Dr Memory reports that there was 2 bytes of leaks, it originates from mingw, and not their own heap members. Therefore, their Snake Game implementation does not cause any memory leaks. This is because their heap members are only allocated in the constructors of their objPosArrayList and Player classes and are then deallocated in their destructors. This means that whenever these classes are called, their destructors automatically deallocate their heap members at the end of their lifetime.

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 really like the use of githubs collaborative work you can achieve using it. It was easy to work together, as we'd just push and pull every time we were done a specific task. This made it so much easier than resending entire code to re copy and paste. The github website itself is very easy to work with as a beginner and is very organized.