COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Angelo Turco, Ryan Calicchia

Team Members Evaluated Junseo, Shajeeven

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 header files displayed offer a comprehensive overlook on how specific elements in the code are made to interact with each other. This is specifically accomplished through the naming in these headers and their accessibility which serves to give an understanding of object functionality, as well as the extent of their referencing. An instance of this great naming can be seen in the food.h header file with public access of functions generateFood and getFoodPos, which both clearly indicate their purpose in both generating food and getting the position of said food. This does not just hold for the publicly named objects however, as in the GameMechanics.h file there are variables within the private accessibility such as int score and char input, which can clearly be seen as variables for incrementing the score of the game as well as retrieving user input As for negatives of what was seen in the header file, nothing too explicit, there is an odd syntax choice which is that they chose to ascribe their private members at the end of the file instead of it being at the start as seen in food.h. However, accessibility is explicitly named so it is not difficult to discern this change in typical usage. Also, in GameMechanics.h int speed is declared, and while this would suggest that speed is kept as an adjustable factor, there is no call within the rest of the code that requires or uses this directly. While this does not directly impact the functionality of the code, it does make it more confusing from an outside point of view. Other than that there are not really any explicit negatives that can be derived from the header file setup, which depicts an overall very intuitive setup.

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.

The main logic in the program loop is easily interpretable, with comments being provided on what particular code blocks are aiming to achieve. The logic is also spaced out very well, allowing for further ease in interpreting the code. The thought-out names for the variables and pointers made it easy to

understand what it was meant for. An example of this can be seen in the initialization of the instances myGM, myFood, and myPlayer which have commented references to the gameboard, food and player. A variable temp then gets assigned the position of myPlayer, which is then given as a reference to myFood for spawning purposes. Nothing in the provided logic calls for questioning about object interaction as well, as they are all very well named allowing for simple interpretation of what they are meant to do. This includes naming choices seen in RunLogic such as UpdatePlayerDir movePlayer and ClearInput which are shown in a sequential order that offers a clear understanding of the movement of the game; update the player direction, move the player based on said direction, and clear input for the next iteration. The print logic of the main function is definitely the most explicitly code heavy portion of the main function which could be potentially spaced out better to allow for perfect legibility. However, this is a small and perhaps nitpicky choice of a negative, as even the spaces throughout the function are done extremely well. Other than this, instances are correctly allocated on the heap and deallocated with good naming convention, the Runlogic portion of the code is extremely easy to understand as well due to the naming clearly showing what should be happening. As well the use of positions in discerning what should be printed is greatly done as well, with sensible naming and explicit object interaction which shows how it is compared and utilized in printing. This can be attributed to reference names such as tempFoodPos and PlayerBody, which clearly indicate the usage of both player and food positions. Overall, the code is extremely sensible in this regard.

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.

The pros and cons for the project and PPA3 in our experience can be described by most of the general characteristics of C++ OOD and C procedural design.

C++ OOD Approach

PROS:

- Modularity makes it easier to manage the given code
- Code is easier to read and understand due to distinct functionality regions
- Code can be reused, eliminating redundancy in certain areas
- Code is easier to debug as area of bug interest is more clear in viewing (specific file for specific functionality)

CONS:

-Code accessibility makes user constantly work to maintain access in different areas of the code

-More complex than simple line by line code

-More difficult to manage over multiple files compared to 1 (PPA3 v project management)

-Code flow can feel more disjointed, having to read multiple files for full understanding

C Procedural Design Approach

PROS:

-Simple coding setup, line by line

-Don't have to worry about referencing other files explicitly (other than standard library and such)

- Can get a very quick understanding of code flow by reading one file

-Easier to work on due to the lack of files and linear structure

CONS:

-Harder to read on average due to large structure

-Have to reuse specific lines of code repeatedly within the code (no form of inheritance)

- More difficult to create large projects due to the amount of code in one place (can be overwhelming and hard to manage)

-Harder to debug due to great amount of verbose statements line after line

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.

Explanations are provided in needed areas throughout the code that explain what that particular region of code is intended to accomplish. Overall it is comprehensive in explaining what each area should do, but does not really explain how the inner mechanisms of said areas work. For instance, someone who has minimal coding experience would not have reference to how the functions actually work, but would understand what they are intended to do. In example, the food generation logic has the comment "for food to not overlap with the snake's body" before then having 9 lines of code after it to derive this logic without explicit explanation. While for our coding group we could then digest this logic to understand the selection process, it did take extra time to do compared to if the most intricate portions were explicitly commented. To improve upon this, more comments could be added to the inner executable lines of function areas so that a more comprehensive understanding of the function could be achieved. In areas with repetitive logic such as food generation or the printing of the board screen, only one of the small blocks could be commented more in depth for a truly quick and easy understanding of the full code to be achieved. By adding even just a comment for count in the food generation logic such as "tracking the amount of food

objects spawned" would increase one's ability to efficiently understand the code given. While this overall commenting is a minor concern, we believe it would improve this aspect of the code.

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 is very legible, and indentations are done so that it can be easily understood what statements are executed under the same conditions. This is explicitly apparent in both the print screen logic and food generation logic, which have all code segments within the same block of execution in line with each other through indentations making it easy to understand the loops. White space is used well overall and increases the legibility of the given code, but more could be used to further increase legibility. This is particularly apparent in the food generation logic where there are plenty of comments and executable statements in subsequent order. While this is still decently legible, it could be improved by having more white spaces in between the verbose areas. Newline is used very well in terms of the on screen in game printing, allowing for good user interface. This is readily apparent in the game movement logic that is printed, as well as the end game logic. Overall, this area of the code is quite strong and overall legibility makes it easy to understand what is going on.

Part III: Quick Functional Evaluation

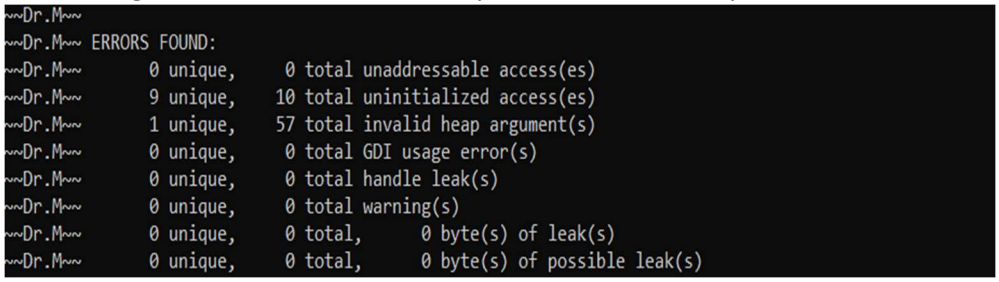
1. **[8 marks]** Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just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

   Overall, the game offers very smooth gameplay that is seemingly bug free. First a line is printed saying Snake Game onto the console, before then printing out the rest of the game board. This has the player icon at the center of the board and other items are printed around it such as the food items, spaces, and # characters which are used to complete the board. Then on the below lines, there is a score counter and then instructions for how the player movement works each on newlines, and then a final line to specify the game termination command. These are all printed onto the screen seamlessly, with the esc function correctly ending the game showing the game over screen. This proves the viability of the print logic in their code. For food generation, 5 items appear on the screen every time with 3 being normal food items ('o') and 2 being special character foods ('S') and ('$') respectively. Collection of the 'o' food inserts a head and increments score in a smooth looking fashion while the while the '$' character simply adds 1 length and increments score by 5, this collection is also done seamlessly. The 'S' increments the score by 1 and increases the snake length by 5 which also appears smooth. These foods also all spawn at random locations, and always prints exactly 5 foods corroborating that the print

collision logic, food generation logic, food collision logic, score counter logic, and increase of player length logic are all sound. The snake movement seems to be perfect, with proper wraparound and head insertion and deletion appearing consistent and logical. Snake collision logic is also correct with an appropriately timed you lose screen upon collision of the snake proving appropriate player and game mechanics functionalities. Overall, in the gameplay experience there appears to be no qualitatively observed bugs and the game runs very smoothly.

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.

   The snake game does not cause memory leak within the analysis as seen below:



   This is due to the fact that all allocated memory throughout the code has its memory deallocated either directly like seen in the project.cpp portion, or through destructors. For instance in project.cpp 3 instances are assigned memory with new (myGM,myFood, myPlayer) in the initialize routine. In the cleanup routine thereafter, all these instances are deallocated with delete statements. In Player.cpp playerPosList = new objPosArrayList is used to track player positions. This is then deleted in the given destructor using the delete functionality. In objectArrayList.cpp aList= new objPos [ARRAY_MAX_CAP] is allocated and then deallocated in the destructor using delete [] to compensate for the array. Finally in food.h, the foodbucket is allocated memory with foodbucket=new objPosArrayList() and then once again deallocated with the destructor. This fully encapsulates all memory assigned in the program, which can be seen as all being deleted which is why no memory leakage is present.

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 pretty enjoyable to work on the project and we think that it was valuable to furthering our learning as engineers who wish to code in an actual workplace setting. We both enjoyed working with one another and found it fun to work on troubleshooting a problem together rather than by ourselves. The communication during the tough stages of the project proved to be very helpful as we could both have input on potential solutions. We think it was extremely beneficial to our learning of not only C++, but also our understanding of what coding as a collective is like. We hit snags on the generation of our foodbucket for the bonus in particular, but after about a day of work and constant collaboration we were able to discern the problem to be due to code accessibility. Overall however, we think that this was a great experience as a coding group.