COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Ali Naqvi & Borna Askarimajdabadkohneh

Team Members Evaluated Amna & Sarah Ahmed

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.

Based on the header files for each object we can make several interpretations about how the program will behave. Starting with Food.h, the line objPosArrayList *foodBucket; suggests food positions are stored in a dynamic array to handle multiple food items generated on the screen. The method void generateFood(objPosArrayList &blockoff); tells us that this method will be responsible to generate food while considering factors such as the position of the player. Another important method in Food.h, is void getFoodPos(objPos &returnPos, int i); which indicated that the program would retrieve the positions of the generated foods to use for other sections of the code such as collision detection with player). The header file GameMechs.h has various methods such as getExitFlagstatus(); setExitTrue(); setWinFlag(); which are methods used to control the game state and to reflect the changes as the game advances.

For Player.h, the observed methods are movePlateer(); updatePlayerDir(); checkFoodConsumption(objPos&currHead); increasePlayerLength(objPos&currHead) all of which have to do with the player movement and its interaction with the generated food. Furthermore, Player.h includes enum Dir {Up, Down, Left, Right, Stop} indicating the possible directions in which the player can move in. When the game is initialized, an instance of GameMechs will be created to initiate the game environment. During the game GameMechs obtains the user input and passes it to the player object in which it is used to update the direction of the player with updatePlayerDir(); and movePlayer();. Player object also interacts with Food by checking if any food has been consumed through checkFoodConsumption(); and if they player head has the same position of a food it will get longer and the score of the player will get updates through gameMechs.

The positive features of the code include the use of private and public members in classes which is regarded as a good coding practice specifically for the additional food class created. The use of various classes responsible for handling different parts of the game is also a good coding practice in OOD. The only possible issue in the header files is that for Food.h and GameMechs.h, they both include each other which can cause issues in terms of circular dependencies which may require forward declaration.

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.

When examining the main logic in the program loop it is easy to interpret how the objects are interacting with each other. Starting from the beginning, some pointers are declared in the global scope that are utilized in the main logic as the program progresses. Looking at the 'Initiailize()' routine, it is easy to interpret how the 'getExitFlag()' object works in order to extract an input from the user. Moreover, it's evident in the 'RunLogic()' and 'DrawScreen()' routines that various objects of different classes work together in tandem to make the program run smoothly. For example, the main screen drawing logic utilizes various objects relating to board size, snake body size, and co-ordinates of food objects to ultimately draw the interactive snake game screen, ensuring boundary control.

These objects are well designed in an OOD manner ultimately making the readability of the main program loop very easy to understand. Overall, the implementation of objects and the way they interact is well done in the main program loop. In terms of negative features, there seem to be none relating to the way objects are handled. Perhaps, adding even more new methods to classes to print messages could be an addition to reduce overcrowding of print statements at the end of the program.

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 design encourages grouping data and using various methods which can be helpful when designing a complex and extensive program
- Object oriented design allows to conceal the internal parts of the code through private members and only display the required public methods which can help protect the overall integrity of the program.
- Object oriented design encourages code reuse as new classes and can use or in other words inherent characteristics and features of previously developed classes making the code more consistent and reducing development time.
- The most notable benefit of Object-Oriented design from my subjective experience is having the flexibility of modifying or altering one part of the code with minimal to no impact on other sections of the program which reduces the risk of bugs and errors.
- The major downside in Object oriented design for me was implementing the wide range of features correctly as a beginner requiring careful design and numerous tests to avoid any issues.
- Another downside to Object oriented design is the use of more memory and increase compilation time however this downside in not noticeable at the current level of object-oriented programing done for the project.
- Procedural programming in C is simple and straightforward making it easier to learn and implement in the PPAs throughout the semester.
- When working on smaller projects such as the PPAs, implementing a procedural program is more practical and time saving.
- Using procedural programming for long and complex projects can become very challenging as it does not allow the use of structuring mechanisms such as classes and objects.

- Procedural design approach makes it more difficult to reuse code in comparison to Object oriented design resulting in more repetitive code in the program.

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 does offer a good number of comments ultimately displaying an excellent self-documenting coding style. Many of the member functions of classes as well as areas in the code where these functions interact with each other have a good amount of commenting and self-documenting that allows the reader to understand the code functionality. Overall, very well done.

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 does a perfect job in following indentation standards and deploying new lines when necessary. This is seen throughout all parts of the code making it easy to follow and understand and improving the overall readability of the program.

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. There seem to be no errors or bugs occurring when playing the game start to finish and the food items as well as the snake body generation are done smoothly and well.

The only potential shortcoming doesn't relate to the actual gameplay experience but the logic of the snake board display and food generation logic. During the generation of 5 food items (inside Food.cpp in generateFood() function), the number of spaces available on the board for food to be generated is hard coded to 18 by 8 (corresponding to the 20 by 10 game board, excluding the border) in a 2D array (see Food.cpp Line 27). Although this works for the 20 by 10 gameboard, if any other dimension of the game board were to be inputted, perhaps a smaller dimension than 20 by 10, then the food generation logic would cause bugs to arise since it has already been hard coded that the available spaced are 18 by 8. For example, if a 15 by 7 gameboard were to be initialized (using the gameMechs constructor in 'Initialize()' routine), during food generation, the logic will assume 18*8 spaces are

available when in reality there are 13*5 spaces available. Because of this, when food items are generated in lines 43 and 44 (inside Food.cpp), there is potential for food items being generated outside the board. This *potential* bug can be fixed by changing the hard coded values of 18*8 in lines 27,43 and 44 into dynamic values that depend on the specified game board size (ex: GameMech->getBoardSizeX() - 2).

In conclusion, the snake game is very well implemented and has *zero* buggy features. However, there is potential for bugs to occur if the specified gameboard dimensions are changed.

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.

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      8 unique,    90 total uninitialized access(es)
      0 unique,     0 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      0 unique,     0 total,      0 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
ERRORS IGNORED:
     20 potential error(s) (suspected false positives)
        (details: C:\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.17960.000\potential_errors.txt)
      1 potential leak(s) (suspected false positives)
        (details: C:\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.17960.000\potential_errors.txt)
     37 unique,    37 total,  10334 byte(s) of still-reachable allocation(s)
        (re-run with "-show_reachable" for details)
Details: C:\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.17960.000\results.txt
```

Looking at the DrMemory report, there seems to be zero memory leakage in the program. All allocated heap members were correctly deallocated to prevent this. For example, in the ‘Initialize()’ routine inside the main program, when ‘myGM’, ‘myFood’ and ‘myPlayer’ were initialized on the heap, their instances were also deallocated in the ‘CleanUp()’ routine using ‘delete’ ensuring no heap members were left on the heap. Additionally, any heap members that were declared inside classes, such as the ‘objArrayList’ pointers inside the constructors for the ‘Food’ class and ‘Array List’ class were also deallocated in their respective destructors using the delete command which would have triggered at the program end. By deallocating all heap members, the code ensures zero memory leakage.

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.

Team collaboration during this final project was a good experience. It allowed us to gain valuable experience in working in a team-oriented environment which is crucial in the industry since many projects are done in teams instead of solo. Additionally, working as a team allowed us to work on the project in chunks in a step-by-step manner ultimately mitigating exhaustion of taking the task alone. Lastly, it taught us how to use git to save snapshots of each other's codes to ensure better workflow.

One thing that may need to be improved on is the usage of git/GitHub in a team environment, specifically how to ensure commits go unchanged and don't overwrite team members codes. Perhaps a crash course on git involving repo branches and other useful tips would have been good.