COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Mehak Shah and Nandha Dileep

Team Members Evaluated Alexander Arruda and Bradley White

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.

After looking through the header files of each object, it can be concluded that the behaviour of the objects involved in the program and how they interact with each other is easily interpretable. The methods and member variables clearly indicate what an object can do and what information it can store. For instance in the Food.h file, this team adds an additional getter called `getAmountToGenerate()` which clearly is used to generate a specific amount of food items in the gameboard for the bonus. The implementation of the additional getter proves that this group understands how a class is used thoroughly. Moreover, the included header files at the top of a header file reveals the dependencies between the objects and how they interact with each other. This can be seen as in the Food.h file, they have global inclusions `"objPos.h", "objPosArrayList.h", "GameMechs.h", "Player.h"`, to include those contents into the Food.h file. Then they effectively use pointers to link the objects to other classes. Looking at all of the header files as a whole, it is very clear to see what the purpose of each class is and how one file can build off of another file.

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 program loop allows for easy interpretation of how the objects interact with each other in the program logic. The interactions between the main objects are considerably clear and well structured. For instance, the `Initialize()` function sets the game up by creating instances of `GameMechs`, `Player`, and `Food` which establishes a clear relationship between the objects. Moreover the use of pointers such as `Game`, `playerPos`, and `foodBin`, provides flexibility in managing their lifetime as it allows dynamic memory allocation. The use of pointers helps pass the objects to functions by reference, which allows for modifications in the functions. However, a negative feature in the main program logic is that it is over engineered. They essentially check the input twice to determine whether the player direction should be changed. In `RunLogic`, they check for specific movement keys and call the method `updatePlayerDir()` for the object `playerPos`. However, `updatePlayerDir()` checks the specific movement keys again. Thus, they should omit the switch statement and just call the method `updatePlayerDir()` for the object `playerPos`.

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.

Procedural Programming:

Pros	Cons
● Simplicity: Procedural code may be simpler and more straightforward, making it easier for developers of all level to understand and develop	● Difficulty in Scaling: As projects grow in size and complexity, procedural code can become harder to manage when new features are required ● Difficulty for Groups: Procedural code may be difficult for large teams of programmers to develop in unison as it requires understanding previously existing functions/code. ○ Thus, the team members are required to understand the entire program fully to correctly implement new code

Object-Oriented Design:

Pros	Cons
● Modularity: Modularity is facilitated by object-oriented design, which enables programmers to encapsulate functionality within objects. This improves code organization and maintainability ● Code Reusability: Through inheritance and polymorphism, object-oriented design encourages code reuse, cutting down on redundancy and program development time. ● Abstraction: A higher level of abstraction is offered by object-oriented design, which makes it easier for developers to model real-world objects ● Encapsulation: Encapsulation helps conceal an object's internal workings, This improves code structure and security	● Learning Curve: Since object-oriented programming requires a strong grasp of inheritance, polymorphism, and other concepts, it may be difficult for beginners to grasp object-oriented concepts ● Difficulty in Design: A well-designed object-oriented system requires careful consideration of the relationships and interactions between classes. A system with poor design choices may be challenging to use and maintain in the future

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 comments present in this code are thorough and well-intentioned. Comments are present in all areas where the code intentions are not obvious to explain what the code does and how it completes the task, as well as explaining where the relevant variables are to be used. This allowed us to understand the intentions of the program and verify the logic/interactions without having to study and deconstruct the code. The programmers also did not over-comment on the code by explaining the roles of very simple functions/methods where the intentions were evident.

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 well organized for readability. Consistent indentation, new line formatting, and whitespaces were all well utilized to create well structured code.

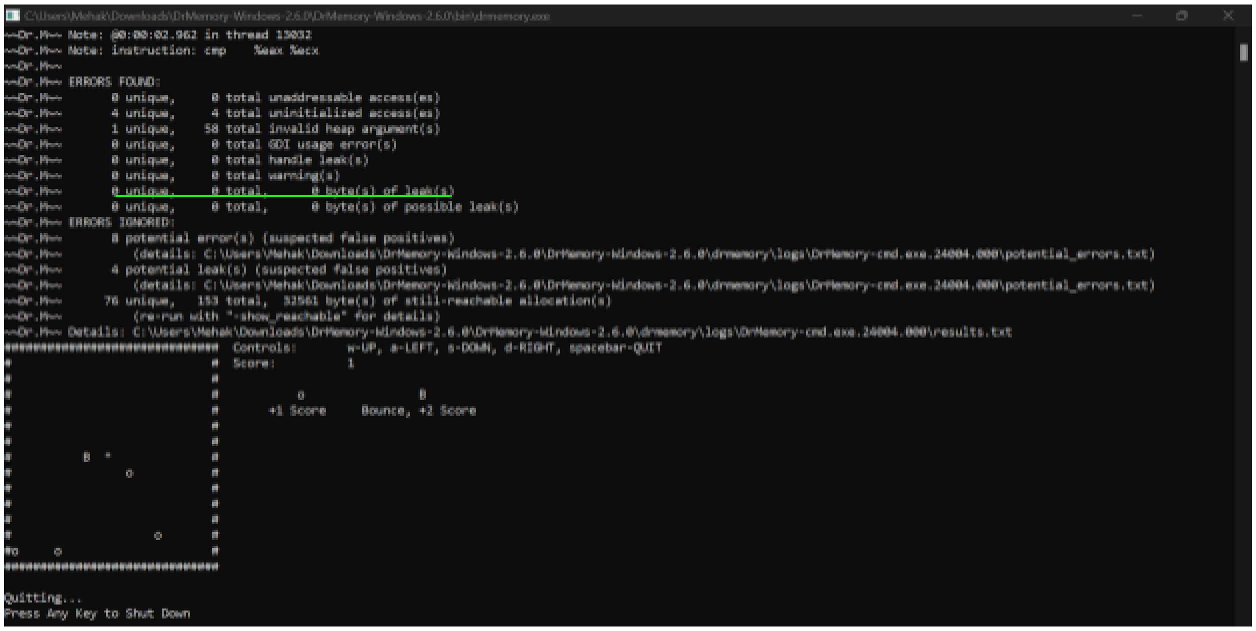
Part III: Quick Functional Evaluation

1. **[8 marks]** Does the Snake Game offer a 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 thorough testing, there were no bugs detected in the Snake Game. The Snake border wraparound is implemented correctly. The escape key "space bar" and the self-collision condition both end the game with a "Quitting…" and a "Game Over + Score: Amount" message respectively. The Snake increase is also done without any errors, as the Snake length increases by one every time a regular 'o' is consumed.

In terms of the bonus condition, five food items are correctly randomly generated in the empty spaces on the board. The 'o' food increases the score and length by one while the bonus 'B' food increases the score by 2, the length by 1 and changes the snake direction. This occurs without any errors and Snake reverses through the board in the same path it traveled to get to the food. The number of 'B' foods vary from 0-2 during each new food spawn.

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.

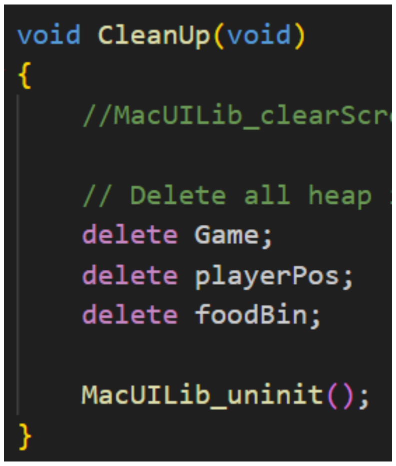


```
C:\Users\Mehak\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\bin\drmemory.exe
~~Dr.M~~ Note: @0:00:02.962 in thread 13032
~~Dr.M~~ Note: instruction: cmp    %eax %ecx
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       4 unique,     4 total uninitialized access(es)
~~Dr.M~~       1 unique,    58 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total GDI usage error(s)
~~Dr.M~~       0 unique,     0 total handle leak(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~       8 potential error(s) (suspected false positives)
~~Dr.M~~          (details: C:\Users\Mehak\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-cmd.exe.24004.000\potential_errors.txt)
~~Dr.M~~       4 potential leak(s) (suspected false positives)
~~Dr.M~~          (details: C:\Users\Mehak\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-cmd.exe.24004.000\potential_errors.txt)
~~Dr.M~~      76 unique,   153 total,  32561 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: C:\Users\Mehak\Downloads\DrMemory-Windows-2.6.0\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-cmd.exe.24004.000\results.txt
##############################  Controls:       w-UP, a-LEFT, s-DOWN, d-RIGHT, spacebar-QUIT
#                            #  Score:          1
#                            #
#                            #         o                B
#                            #     +1 Score     Bounce, +2 Score
#                            #
#                            #
#          B  *              #
#                o           #
#                            #
#                            #
#                            #
#                    o       #
#o     o                     #
##############################

Quitting...
Press Any Key to Shut Down
```

Figure 1. Screenshot of Dr.Memory Report

The Snake Game does not contain any memory leakage. This means the programmers utilized the delete[] commands in the correct locations, as seen below in the main Project.cpp file, as well as in the objPosArrayList.cpp, Player.cpp, GameMechs.cpp and Food.cpp destructors.



```
void CleanUp(void)
{
    //MacUILib_clearScr

    // Delete all heap 
    delete Game;
    delete playerPos;
    delete foodBin;

    MacUILib_uninit();
}
```

Figure 2. Screenshot of Project.cpp heap deallocation

However, it contains an invalid heap argument. This could be potentially caused by erasing the same memory location from the heap twice.

<u>Part IV: Your Own Collaboration Experience (Ungraded)</u>

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.

Our software development project was characterized by strong communication and effective time management. Constant and clear communication between the both of us allowed us to work efficiently, ensuring both team members were on the same page throughout the first two iterations. Additionally, the coordination facilitated the seamless integration of our code into Github without any issues. While the project did progress well, we did encounter a challenge in completing the bonus feature within the set timeframe. Although we decided not to overly stress about the bonus, a more structured approach could enhance our efficiency while doing the bonus. In future instances involving bonus features, we both should strategically brainstorm ideas on how to implement the code and split tasks between the both of us.