COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members	yakubup	hamadeb
Team Members Evaluated	neelamas	sidhum38

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **30 marks**. Do not exceed 2 paragraphs per question.

Peer Code Review: OOD Quality

1. **[3 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 is implemented in Project.cpp. It consists of a while loop that calls the functions GetInput, RunLogic, DrawScreen, and LoopDelay. The logic is modular and easy to follow with each function having it's own separate responsibility. GetInput() handles the user input by checking what key has been pressed by the user every loop iteration. RunLogic() handles game mechanics like player movement and food collision. It also checks if the lose or exit conditions are true every loop iteration. DrawScreen() focuses on rendering the game and other relevant information like the score and controls on the terminal. LoopDelay() ensures consistent pacing of the loop and ensures that it isn't rapidly occurring otherwise mechanics like player movement and displaying messages on the terminal wouldn't work ideally. Again, the main method is very clean and isn't cluttered with variables and logic that can be placed in other functions.

The objects (myPlayer, myFood, and myGM) interact appropriately with each of these functions to execute the game logic. Player and Food have a dependency on GameMechs which also promotes modularity because it allows them to reference GameMechs for game state information. It is very easy to follow along and identify what each function is doing with descriptive names of the accessor and the mutator methods even if the comments haven't added much depth to understanding the code. The use of global pointers could cause issues if they aren't handled correctly, but that is definitely up to the designer.

2. **[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 pros of the C++ object oriented design approach include encapsulation, abstraction, reusability, and scalability. All of the game components like Food, Player, and GameMechs have well-defined boundaries. This provides a clear and controlled interface for interacting

with the class and ultimately promotes abstraction. For example, the score is only able to be incremented through the incrementScore() method, preventing any accidental or invalid modifications. The code becomes much more readable and it becomes clear where the score is being incremented in the code making debugging much easier. The best part about C++ is our ability to reuse our code, especially when developing large scale projects like a game. The classes and methods can be used over and over again without having to rewrite the same code. The code also becomes much more scalable in the sense that new features can be added without major refactoring. Some of the cons of the C++ object oriented design approach include more complex memory management, and coupling. It becomes harder to find memory leaks in C++ as projects begin to scale with classes. You must keep track of every new and delete in multiple different files. Coupling, which is the dependence of one class to another, promotes lower modularity of the code and eliminates one of the biggest advantages of C++ object oriented design.

The pros of C procedural design include it's simplicity and efficiency. Procedural code is more straightforward, and reduces the development time for smaller projects where you don't have to use classes. The C compiler is more efficient too because it doesn't have to consider complex features like inheritance. The cons of C include maintenance, readability, and reusability. It's harder to debug because of the lack of encapsulation and reading the code is much harder because it's often cluttered in large functions. There will be lots of duplication too.

Peer Code Review: Code Quality

1. **[3 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 files contain some comments explaining the purpose of functions and key logic, which is helpful to understand the code. However, some of the comments don't make a lot of sense, are incomplete, redundant or outdated. In some comments, the developers leave notes to themselves or each other which simply clutters the code and doesn't help us understand it any further. It's better to communicate where the code is wrong over text instead of leaving a note as a comment I find. There's a lot of commented code that should've just been removed before the final submission too like in Project.cpp and Food.cpp. I recommend having a comment on top of each function or class to describe its purpose and then comments beside important lines of code that describe the logic which was done well in some parts. The naming conventions are clear everywhere and it's easy to understand the purpose of different classes, functions, and variables. The self-documenting style observed by the developers was enough to understand the code which was well done!

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.

The code uses proper indentation in all files, making it easy to follow and understand the logic in functions, loops, and conditional statements. Implementing sensible white space in most files could've been done a little better. Some files like objPosArrayList.cpp lack consistent spacing in block structures. Line breaks between logical sections are inconsistent, especially in larger methods like DrawScreen(). Again, there are a lot of unnecessary comments that should've been removed to make the final submission look more polished and professional. I suggest developing consistent spacing and indentation rules, you can choose your preferred step size, whether its 2 lines or 4 lines. I also suggest adding line breaks between different sections of long methods to improve readability.

Peer Code Review: Quick Functional Evaluation

1. **[3 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 technical user feedback)

   The Snake Game doesn't offer a smooth, bug-free playing experience. There are a couple of buggy features. There seems to be a problem with the rendered display's response to the input, with the players actions registering on the next frame when the display should react immediately. Food generation will overlap with the snake's body. This is because the generateFood method only checks to see the location of the snake's head on the board to avoid overlaps. This is improper implementation and should be done by iterating through each element of the snake in the objPosArrayList object and make sure all those positions on the board are avoided. Collision detection also seems to be implemented incorrectly. The checkSelfCollision function could potentially fail if the new head position is added to the list before checking for overlaps, which could lead to missed self-collisions. Addressing these issues involves optimizing the input polling mechanism and reducing delay constants for smoother responsiveness, modifying generateFood to ensure no overlap with any part of the player's body, and reordering operations in checkSelfCollision to perform checks before updating the player's position.

   The lag in input responsiveness seems to be from an incorrectly set delay constant in the main game loop, causing excessive delays between frames. The delay seems to be changed to different values in several different places in the main project loop. It would be wise to trace the logic of these if statements and see where it could have been messed up. One of the conditions could be true even if that is not the desired output. For food

generation, ensure the generateFood method iterates through the entire playerPosList to verify the generated food does not overlap with any part of the player's body, logging positions for clarity using the debugger. Finally, for collision detection, reorder the logic in checkSelfCollision to perform self-collision checks before updating the player's head position, and log positions of the head and body segments to validate proper detection during gameplay.

2. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.

   The Snake Game did not cause any sort of memory leakage. There are no memory leaks in the provided code because the dynamic memory allocations are properly managed with corresponding delete calls in each file. The Dr. Memory reported 0 bytes of leaks and possible leaks, confirming that there are no memory leaks.

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      7 unique,    71 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:
     18 potential error(s) (suspected false positives)
         (details: C:\Drmemory\drmemory\logs\DrMemory-Project.exe.15712.000\potential_errors.txt)
     28 unique,    28 total,   7338 byte(s) of still-reachable allocation(s)
         (re-run with "-show_reachable" for details)
Details: C:\Drmemory\drmemory\logs\DrMemory-Project.exe.15712.000\results.txt
```

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to yours, reflect on the following questions:

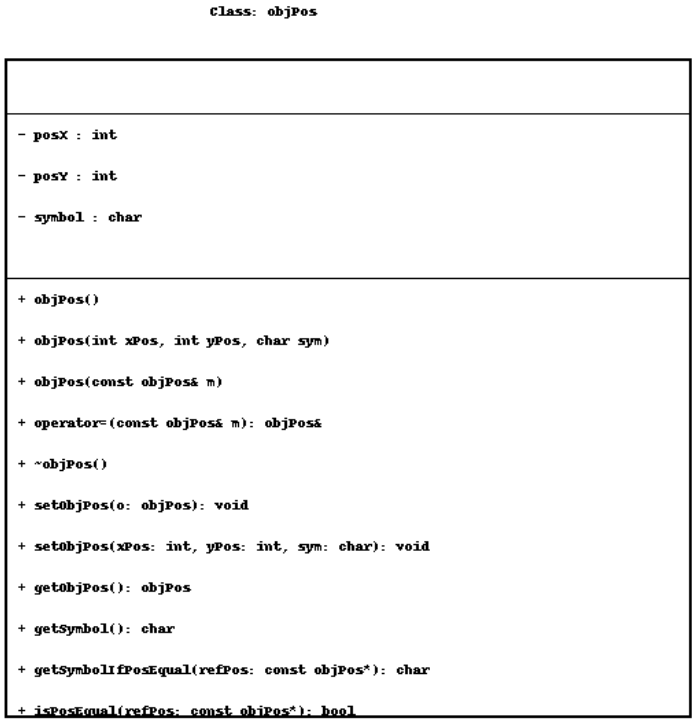
1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

   The compound object design of the objPos class is functional but has unnecessary complexity. Using a pointer for Pos (Pos* pos) is unnecessary, as Pos is a small struct and could be directly stored as a member variable. The dynamic allocation of Pos introduces risks like memory leaks if the destructor, copy constructor, or assignment operator are implemented incorrectly and it unnecessarily complicates the design. The class makes progress toward the minimum of four/rule of six by including a copy

constructor, assignment operator, and destructor. However, to improve we can replace Pos* pos with a direct Pos member or the position can be stored directly as posX and posY as private variables in the objPos class which would streamline the implementation. This change would make the class cleaner, more efficient, easier to maintain and more aligned with C++ object oriented design principles.

2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. <u>You are expected to facilitate the discussion with UML diagram.</u>

   To improve the design, I suggest making two new private variables to implement the posX and posY since that is the main function of the struct. There is no point of having a separate holder for the variables. Here is a UML diagram showing it.



This diagram shows the posX and posY variables as private variables of the objPos class. The UML diagram proposes an improved design for the objPos class that better aligns with C++ object oriented design. It effectively eliminates the unnecessary complexity of the original implementation, which was caused by the dynamically allocated Pos struct beforehand. It uses direct member variables posX and posY to represent the coordinates simplifying the design and improving maintainability. This change removes the extraneous dynamic memory

allocation for a struct, reducing the risk of memory leaks. The methods, such as setObjPos, getObjPos, and isPosEqual remain intuitive and operate directly on the simplified member variables, also making the class more efficient and easier to use. Ultimately, this design avoids pointer dereferencing overhead, streamlines memory management, and offers better alignment with modern C++ object oriented design.