COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members alhindm [candels-fall2024 (git name) candels (macID)]
Team Name: iBio Dream Team

Team Members Evaluated baggad1 patem221
Team Name: Codelore

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.

   After examining the main logic in their main program loop, we have observed that their object initializations and logic all correctly interact with each other, though lacking in sufficient commenting to explain their algorithms. All created objects are defined, allocated in the initialization routine as needed, and used in functions like ***RunLogic()*** and ***DrawScreen()*** to check for collisions with the snake's body or food items. For example, food generating depends on the snake's position, and collision detection requires accessing the snake's state, all included in their code but as mentioned, lacking sufficient commenting. Class-specific logic like ***snakeFood*** and ***game*** are handled within their classes, reducing large pieces of code from flooding the main program file. However, we have also observed a lot of code that has been commented out but not deleted and some arbitrary testing values still running in their code which affects the readability and quality of their code.

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.

   Pros and Cons of C++ OOD approach

   Pros

   - Encapsulation of data, and logic-specific functions are grouped together in their respective classes for readability of the main program.
   - Modularity and separation of code – having many classes like Player, GameMechs, Food, etc. helped organize and handle distinct functionalities of the different moving pieces. (C in PPA made us define our random generation function in the main c file.

- Inheritance of classes to allow for more reusability, we can easily create and extend new classes to continuously improve and upgrade our game without rewriting everything.

Cons

- Much more complex - while having many organized classes is good, it made the project a lot more complex and harder to initially wrap our heads around.
- Having to keep track of all methods, private members, and their data types requires great attention to detail ex. Player class contains a pointer ***Pos*** that points to x and y elements which adds an extra layer of methods to access when getting the player's position.
- As we deal with more pointers and variables, we required more Dynamic Memory Allocation to optimize memory usage which made the code vulnerable to memory leaks.

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.

Throughout the files there was not sufficient comments, and the self documenting coding style was lacking and quite confusing. There were some specific sections where the commenting was excellent for example in the ***CleanUp()*** function of the project.cpp file every if/else-if code block had a comment preceding it and explaining what each block was going to execute. But for the majority of the other files, the comments lacked. Most of the comments made throughout the files seemed to just be old lines of code, or lines used for testing different features, so looking at the comments did not give context as to what the code was producing at these given points.

In general, the commenting lacked throughout all the files so if I had to improve this, I would suggest every 3-4 lines adding in some comments to explain what is happening in the code. 3-4 lines is just a general suggestion but at the very least each code block (loops, conditionals, declarations, etc.) should have a comment explaining what will be executed in the proceeding lines. Particularly, in the ***Food.cpp***, ***GameMechs.cpp***, ***Player.cpp***, ***ObjPos.cpp***, and ***ObjPosArrayList.cpp*** files, it would have been nice to see comment lines above the member functions. As we also completed the project, it was easy for us to follow along and understand the member functions, but the goal of self-documenting coding should be that a programmer who has never seen the files before can understand what the code is doing. Finally, a specific suggestion we make is to include more in-depth comments in the ***Food.cpp*** file and for the collision logic in the ***Player.cpp*** file. From group to group, these specific sections vary greatly because there are different ways to produce the game (especially in Food classes as these were not even a requirement). To improve this we would have definitely made more comments in the ***generateFood()*** member function in the Food class because this is a large code block that integrates seeding random number generation and lots of local variables and OOD methods. Specific comments that explain what each local variable is representing, what conditions are being checked for the while and for loops and then what object methods are being executed and called based on the conditions would have helped organize the very lengthy code-heavy section of the file.

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 coders did an excellent job in ensuring indentation, sensible white spaces and deploying newline formatting for overall great readability. In all the files there was at least one and usually more than one white space between different code blocks. This overall made the files less overwhelming and easier to read, and we appreciated that we could examine and digest one block of code at a time without being intimidating by long continuous sections of code. Indentation was always used excellently, but sometimes the newline formatting was not utilized to the best of its abilities. We examined that in the ***Project.cpp*** file, specifically the ***DrawScreen()*** logic section, there were cases of embedded for-loops and if-else states implemented within the embedded for-loops. While all the indentation was excellent, in this section new-line formatting was lacking and the code's readability was not great. Its our preference to place the ' { ' on a new line below the for/if/else/etc. statement and once a ' } ' is placed, implement sensible white space to break up the code.

Also, in terms of overall readability, as we mentioned in question 1, there was often comments of old code that had been updated or old lines that were used to test code functionality. While we understand it is convenient to keep those comments there for debugging purposes when upgrading the code, it often added unnecessary clutter to the files that was overwhelming to look at and made the code readability worse. We would suggest removing all these unnecessary lines of code in future final copies of projects.

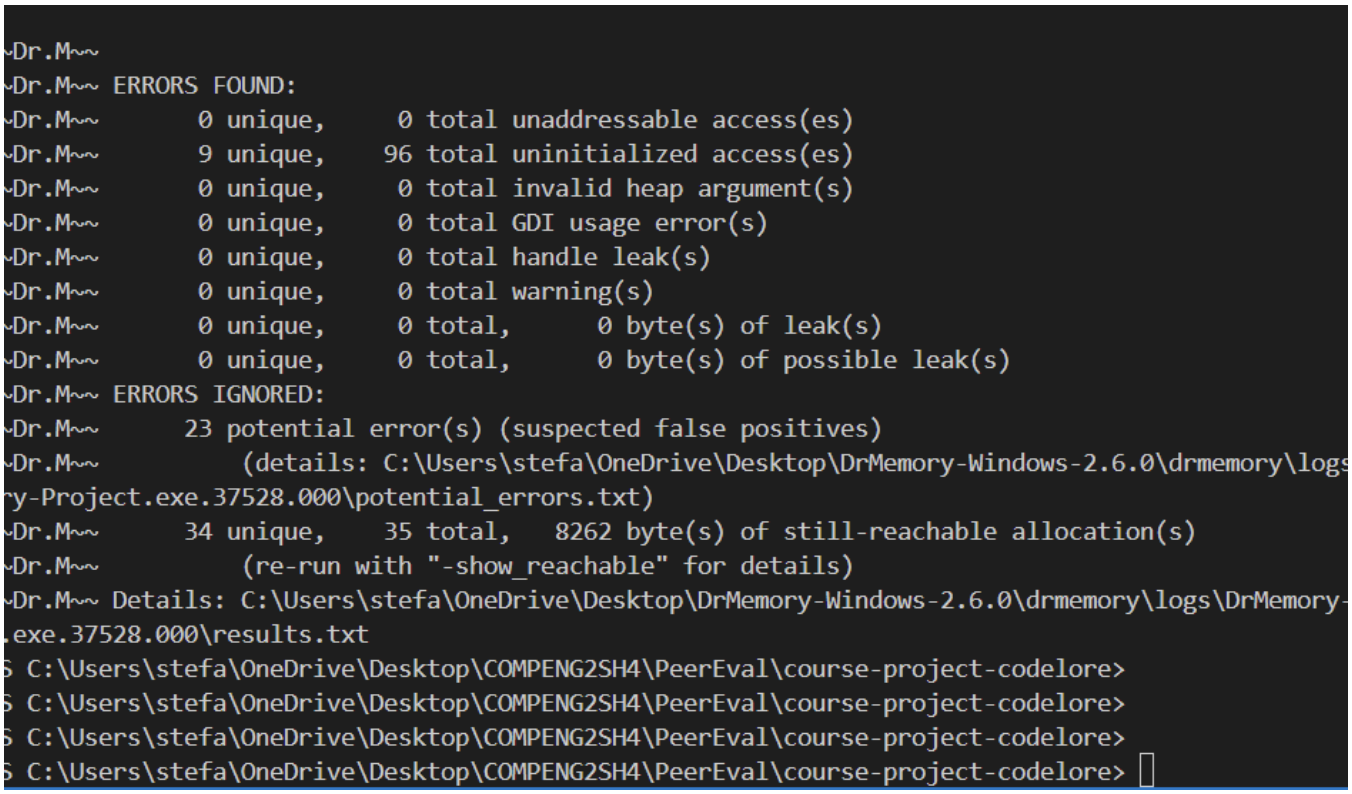
Peer Code Review: Quick Functional Evaluation

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)

Yes, the game ran smoothly and bug free. We tested both the horizontal and vertical border wraparound logic and there were no errors or buggy movements of the snake. We played the game multiple times and were able to reach over thirty points with absolutely no issues. The snake growth and food-collision/regeneration logic were perfected, the snake grew one unit every time the head collided with a food item and the food item was immediately regenerated to a new random spot on the game board not occupied by the snake. We appreciated how the current direction of the snake was displayed, but we would suggest displaying a message that indicated to the user how to exit the game as that was unclear. Once we figured out the exit button was the ESC button, we were able to confirm different messages were displayed when the game was exited versus when the game was lost. Overall, the user experience was very smooth with seemingly no logic bugs to the game, we just had some suggestions to improve user experience that did not relate to the logic behind the actual snake game.

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.

No, we called “drmemory ./Project.exe” in the terminal and produced a memory report of the program and it was determined and observed that no memory leakage was experienced. These coders did an excellent job ensuring all allocated heap memory was sufficiently (and properly) deleted in their ***CleanUp()*** function. This was observed throughout all the files, every time “new” was called to allocate heap memory, there was a proceeding “delete” call to free that memory once it was not needed anymore.



As seen in the above Dr Memory report, zero bytes of leakages were detected.

Project Reflection

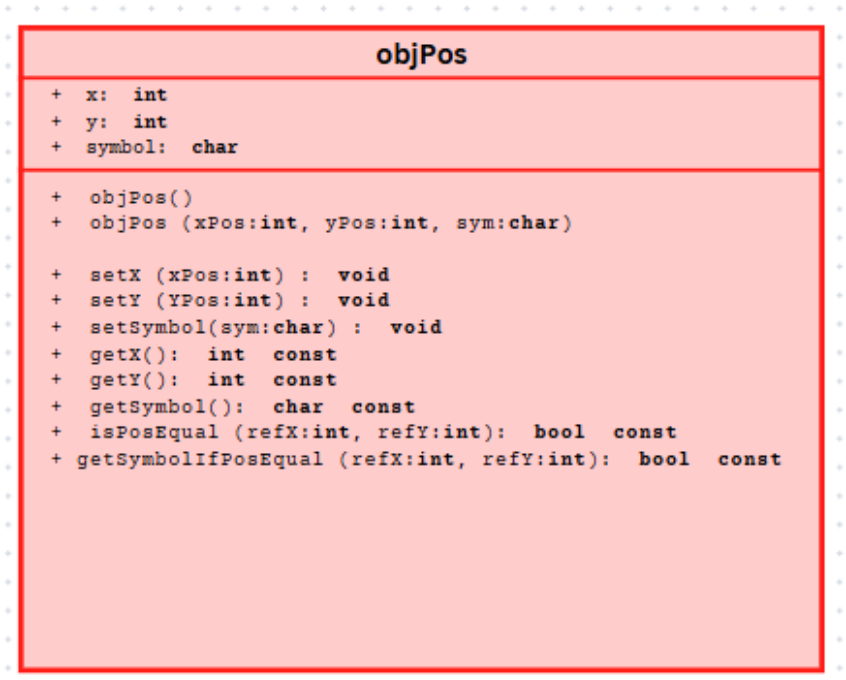
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to yours, reflect on the following questions:

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

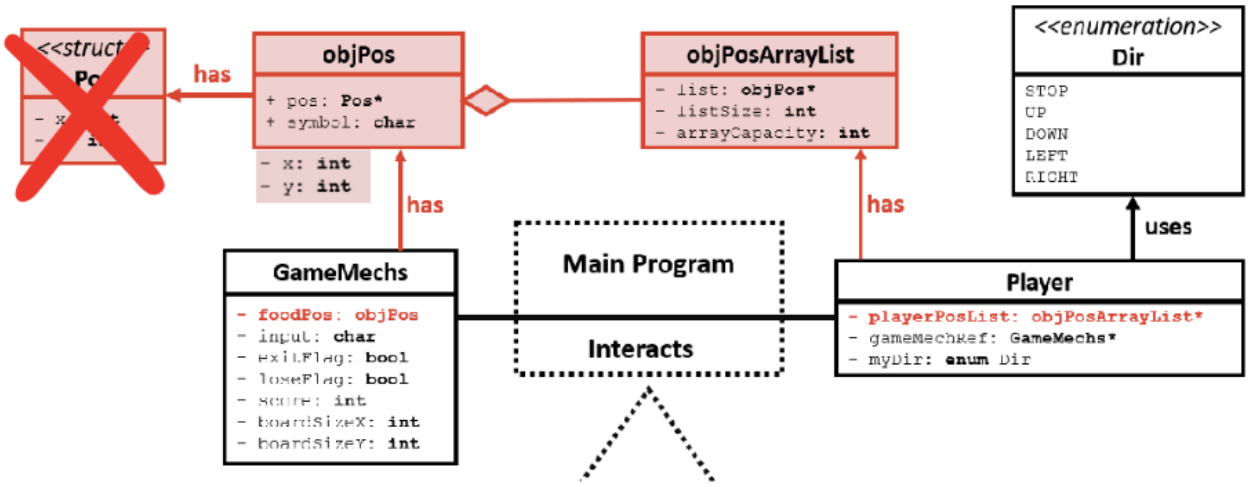
We believe although including the additional Pos struct might have some benefits like position data separation, it is not a sensible design choice since it encapsulates necessary variables (x and y positions) in a separate pointer that requires definition and invoking every time. The ***Pos* pos*** pointer introduces an additional layer of complexity and potential to run into memory issues especially when the ***objPos*** class could hold x and y as private member variables. This was especially seen when attempting to access the snake body's position for collisions where 3 layers of pointer methods were required. This was also seen when trying to access the specific elements of the food bucket, we called "foodBucket->getElement(i).pos->x"; the compound object design added complexity and increased the risk of errors or improper member accessing that we think was insensible.

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. You are expected to facilitate the discussion with UML diagram.

To improve the object's design, we would eliminate the ***Pos*** struct and directly store x and y as integer private members of objPos. Then, getter and setter methods could be used to access x and y just like they're already used to access the symbol. This eliminates the need for dynamic memory allocation as no pointer is used and we can directly interact with x and y positions without a nested ***Pos*** struct.



As depicted by this updated UML diagram the x and y coordinates, which simply represent the objects position on the game board are incorporated into the actual ObjPos class.



Incorporating all of the classes and files this is how the objPos class would interact with the rest of the program. Virtually nothing changes except that when we are now accessing the x and y coordinates of objects of the objPos class we can just simply use the dot operator on the object name. For example in the DrawScreen() of the current program we have to access the x and y coordinates of our object *snakebody* (of objPos class) by calling “snakebody.pos->x”, but the new implementation would allow us to call “snakebody.x” and be able to access the x coordinate the same.