COMPENG 2SH4 Project – Peer Evaluation [30 Marks]

Your Team Members Jyotiraditya Pendyala, Andrew Chai

Team Members Evaluated Julian DeVries, Sam Wauchope

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 logic in the main program loop is modular and easy to follow while adhering to the object-oriented approach. The variable names are also very intuitive and can be understood easily. The less intuitive lines of code, such as print statements for special symbols, are explained using comments, making the code readable and easy to interpret.
   There are, however, a few points of criticism. First, while the delay constant was defined as 50000 µs, the loop delay function mentions 0.1s, which may cause some confusion to people unfamiliar with the code. Second, in the CleanUp() function, the pointers called on the heap are deleted directly instead of calling in their respective classes' destructors (this is more procedural than object-oriented). In addition, food consumption and exit key checks are occurring within the RunLogic() function instead of their classes' member functions, once again breaking the modularity of object-oriented code and reverting to procedural. Lastly, from a player's perspective, the game does not give a message when exited without dying and does not have a guide on the snake foods, making the game less playable. All in all, these are only minor observations, and their code works well overall!

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 of C++ OOD approach:
  - Encapsulation: Classes allowed us to bundle data and behaviours, making the code less dependent on a scattered range of global variables and much more readable.
  - Easy to expand on: Features like the snake body growth and special foods were easier to extend due to the modularity of the code.
  - Reusable: One class could be reused for multiple purposes. For example, we used the objPosArrayList class for both the snake and the food bucket.
  - More manageable code: Due to distinct separation in classes and their associated behaviours, it is easier to manage individual code blocks, update old code, and debug erroneous code without having to change the entire project.

- Cons of C++ OOD approach:
  - More complex: With all the classes, member functions, and the interplay between various objects, it is a lot harder on a cognitive level to keep track of all aspects of the project in contrast to the linear design of the procedural code used in PPA 3.
  - More dependencies: The interplay between different objects can also introduce dependencies, such as the dependence of the player class on the gamemechs class. Such dependencies can hinder modular changes specific to these classes. In contrast, there are few dependencies in the PPA 3 procedural code.
  - Overengineering solutions: Even small issues like handling simple game mechanics are a lot more complex with having to create a separate class with its own unique member functions, whereas procedural code just requires a few global variables in simple functions to emulate similar behaviour.

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.

   Many sections of the code are commented on sufficiently with enough information given to the reader, which allows them to understand what each line should do and its purpose. Be that as it may, there are many missing comments for both header and .cpp files that may not require any comments only if people with coding experience see them. However, to people unfamiliar with the code, there will be much confusion within the code for various member functions. The best way to improve this issue is to remember to include comments for the overall purpose of each function, like writing a very brief description of what the constructor does within objPos.cpp, and explaining complex logic operations, such as the copy assignment operator within objPosArrayList.cpp. Additionally, as explained before, the delay constant was defined as 50000 µs, but the loop delay function has a comment mentioning 0.1s, which confuses the readers. Therefore, it would be best to quadruple-check the comments to make sure no minor errors are made.

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 follows indentation, adds sensible white spaces, and even deploys newline formatting for better readability. The code appeals to the reader and looks simple and precise without too much or too little white space. Each line of code seems to be appropriately indented, within its correct spot, and each statement, noticeably, switch cases, are organized beautifully and synchronously. However, some areas in the code would be much more readable with a new line between two code blocks to differentiate them. For example, the switch statement in the updatePlayerDir() member function in the player class could have a space between each case, easily distinguishing the different cases. But again, these are just minor issues that can be resolved and do not affect readability too much.

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)

   As explained previously, the user-friendliness of the game can be modified more efficiently as no message is given when exiting without dying, and the game does not have a guide on the special food objects. However, the snake game, for the most part, offers a smooth, bug-free playing experience. The snake game functions as it should, with each iteration of the project completed without issues, in addition to the bonus. However, there is one noticeable, minor issue when playing the game: as the snake eats a food object, its body will grow. However, the time it needs to grow is not instant and rather takes a tad bit shorter than a second. Due to this small instance of change, it is easy to miss this bug. Although this issue does not seem to affect the game negatively, I believe it would be best to debug this issue by using the debug method and adding a breakpoint on the line where it involves the snake touching the food object. The issue likely stems from the order of logical operations being performed in the RunLogic() function of the main loop. Food consumption is checked after player direction updates and movement. So, if the snake 'eats' a food, it only grows after it has moved when the consumption is processed. Another minor issue from a player's perspective is that sometimes it is not evident that the player has lost (for example, when the snake head barely touches the tail). For such cases, instead of clearing the screen immediately and outputting the message, it would be better to keep the image of when the user's snake died by removing the clear screen command in the CleanUp() function in the main project file.

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.

   Using drmemory, there was no observed memory leak within the game. Each new variable created through dynamic memory allocation has been destroyed from the heap in the CleanUp() function at the end of the main game loop using the delete keyword. However, as mentioned before, the code would be more OOD-aligned if the destructors were called instead of using delete directly. Nonetheless, there are no issues from a memory perspective.

Project Reflection

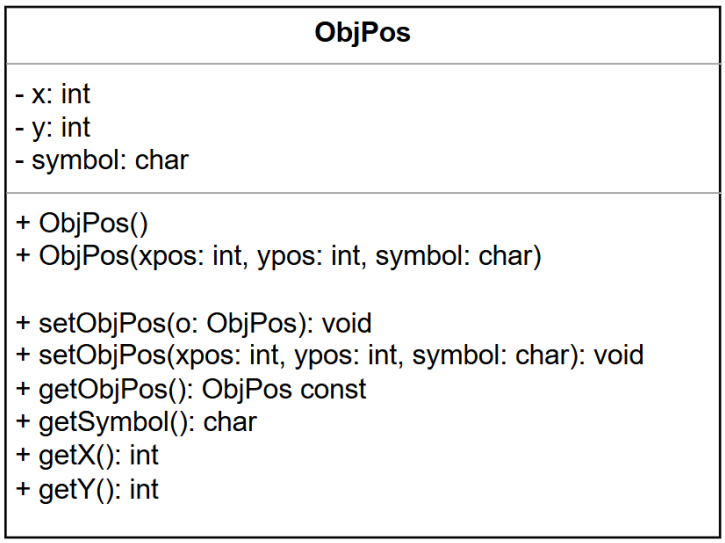
Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to yours, reflect on the following questions:

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

   While it works, the current implementation of the objPos class is not very effective. It uses a dynamically allocated pos struct, creating unnecessary complexity and risk of memory leaks. Even without memory leaks, the frequent instantiation and deletion of the pos struct could further degrade the performance of the code. While it is not noticeable in our case due to the project being very small-scale, large-scale use cases would suffer quite a lot in performance and speed. Instead, it would be a lot easier for the user to embed the pos struct directly into the obPos class, simplifying memory management, improving performance, and reducing cognitive load on the programmers.

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.

   As mentioned earlier, an alternative objPos class design, would involve directly embedding the data stored in the pos struct into the objPos class as private members. This is because the objPos class is very simple, storing just the x and y coordinates and a symbol. Additional modularization by having a separate struct holding the coordinates for such a simple class is unnecessary. The new implementation of the objPos class would simplify memory management by eliminating the need for new or delete calls, improving performance with stack allocation, and ultimately leading to a straightforward and cleaner code without unnecessary segmentation. An updated UML diagram of the class would look something like this:



   Of course, this is a very rough idea of a UML diagram and would need more public member functions such as a copy constructor, equality checker, mutators and other methods that can be added as seen fit for the code.