## 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.**

The header files have all the variable names (public and private) with the functions. Every variable has a sensible name and they all make sense in their respective scope. Possible improvements could be to remove some template comments (ex "`Upgrade this in iteration 3.`" in Player.h to keep the code more concise. Adding comments to explain the functions would be an improvement as it is not immediately clear how something like GameMechs interacts with the Player class.

**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.**

A strong aspect of their code was having an array to store the board and its positions. It helps make the code more adaptable to changes in size and make the drawing of the board quite intuitive. Some flaws are the lack of comments and a large amount of variables that are hard to keep track of. With the lack of comments, while some parts of the code may be easy to understand, comments still make a large difference in trying to grasp what is taking place for each part of every function. At the top of the main file are several global variables which can be difficult to manage and something that should be avoided in OOD. Instead, they should rely on the capabilities of the classes they created and have as very few global variables as possible. Avoiding these mistakes is what could make the code much more readable and functional. Despite this, it is quite easy to tell how many of these objects interact with each other due to well done naming conventions. The clean up of all the declared objects was also well done making sure no memory leaks were occurred.

**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.**

| Pros of OOD | Cons of OOD |
| --- | --- |
| ● Encapsulation helps keep connected properties and attributes together as opposed to many globals all over the place<br>● Abstraction allows multiple people to work on different parts of the same code base at the same time while being very reusable | ● Can result in more complex code as opposed to the procedural approach since you have everything already defined, you don't need to call a function to get those values<br>● Difficult to use relative to procedural designs |

## 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 mostly incorporates meaningful function and variable names, which helps understanding the overall structure and purpose of the code. However, there are still areas where additional comments could enhance clarity, especially in complex or lengthy code blocks since no additional comments have been made. While the function and variable names provide a good level of self-documentation, there are sections that might benefit from more explicit explanations. For example in the GameMechs.cpp file the generate food function shown in *figure 1* utilizes zero comments.

```
void GameMechs::generateFood(objPosArrayList* blockOff)
{

    srand(time(NULL));

    foodPos.symbol = 'O';

    int unique = 0;

    foodPos.x = (rand() % (boardSizeX - 3)) + 1;
    foodPos.y = (rand() % (boardSizeY - 2)) + 1;

    do
    {
        unique = 0;

        for(int i = 0; i < blockOff->getSize(); i++)
        {
            blockOff->getElement(player, i);

            if(player.x == foodPos.x && player.y == foodPos.y)
            {
                foodPos.x = (rand() % (boardSizeX - 3)) + 1;
                foodPos.y = (rand() % (boardSizeY - 2)) + 1;

            }else
            {
                unique += 1;
            }

        }

    }while(unique != blockOff->getSize());

}
```

*Figure 1: Screen Capture of the generateFood function in GameMechs.cpp*

Similarly the comments in the Project.cpp file do not help clarify what the code does; it simply sections out the code. What I would suggest is to look for large code blocks like in *figure 1* and simply explain what they are doing at a high level and explain what certain functions do using in-line comments to help refresh memory.

**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 generally follows good indentation, employs sensible white spaces, and uses new line formatting, contributing to overall readability. The use of consistent indentation enhances the visual structure and appeal of the code, and the placement of white spaces between operators, keywords, and around control structures adheres to common coding conventions. For instance, in the Initialize function in *figure 2*, the allocation of memory for the gridScreen variable is well-formatted, making it easy to distinguish individual components:

```
1  //Initialize Gridscreen object array to the heap ====================
2
3      gridScreen = new int*[GameMechRef->getBoardSizeX() + 2];
4
5      for(int i = 0; i < GameMechRef->getBoardSizeX() + 2; i++)
6      {
7          gridScreen[i] = new int[GameMechRef->getBoardSizeY()];
8      }
9  //====================================================================
```

*Figure 2: Screen Capture of the initialize function in Project.cpp*

## 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 COMP ENG 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 thoroughly testing the Snake Game by intentionally trying to break it and exploring potential issues with wraparound and tail growth, I'm happy to report that the game demonstrated a smooth and bug-free playing experience. I intentionally pushed the game to its limits by manipulating user input and attempting various scenarios, but the game behaved identically to the sample executable. The wraparound feature and tail growth were handled seamlessly without any glitches or unexpected behavior. The code appears to be well-implemented, and the game's functionality aligns with the expected behavior.

**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.**

The Dr. Memory report does not explicitly mention any memory leaks in the final summary. The "ERRORS FOUND" section indicates no issues related to memory leaks, with the "0 unique, 0 total, 0 byte(s) of leak(s)" entry. The destructors are well-implemented, ensuring the “freeing” of allocated memory on the heap without unnecessary deletions. The team adheres to correct C++ syntax in their destructor implementations.

## 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**

We think the project overall was well structured and helped strengthen our abilities in different types of programming. While the coding part of the project was good, there were some flaws we both found. Using git was definitely a big challenge, we faced several issues when trying to pull and push code and we believe that a few classes on git/github would have gone a long way. We were dealing with merge conflicts, stashing changes, and a few other parts that made the experience more frustrating, but overall the project was well executed.