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 behaviors 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.

In this project, the header files are named well, giving a good idea of what each class is about. When you check the names of the set methods in each class, you can figure out what the class is meant for and get an idea of what each method does. For example, in the objPosArrayList.h file, each method name suggests that this object is about working with arrays.

In the project.cpp file, most of the organization is done well. A good example is myFood->generateFood(myPlayer->getPlayerPos()), which tells you that myFood is creating food based on where the player is. The only downside is the myGM pointer name in project.cpp because using acronyms like this might be confusing. For example, myGM->getExitFlagStatus() seems to be about exitFlag, but if you forget what the acronym means (GameMechanics), it could be hard to understand.

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 majority of the object interactions are encapsulated within their own methods which encourage information hiding. Each class is responsible for its own behavior contributing to a modular and maintainable codebase. This also improves code organization and reusability. For example, the player (myPlayer), game mechanics (myGM), and food (myFood) classes collaborate to handle user input, update game logic, and render the game screen. The interactions are appropriately designed to maintain a clear separation of responsibilities among the objects. However, for example, there might be an implication of tight coupling between the ‘Player’ and ‘GameMechs’ classes due to their dependencies which could play an impact when trying to extend the game to add new features.

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:

- **Encapsulation:** prevents certain data members or functions from being accessed by the user - to allow for code security and error reduction in case the user overwrites an important data member.
- **Modularity:** Code is organized into classes with their own purpose improving maintainability and extensibility.

- **Reusability:** Classes can be used in other parts of the program or even in other programs by using Inheritance/ Composition.

Cons:

- **Complex Interpretation During Git Pulling:**
    - Recalling every method in specific classes created by my partner became a challenging aspect during the git pulling(version control stage).
    - Instances where I forgot the involved classes necessitated tracing back to the root class to comprehend the interaction flow.
- **Challenges with Object Names and File Complexity:**
    - The multitude of distinct object names adds a layer of difficulty in understanding the code.
    - The overall complexity of each header file is a further challenge, especially when referencing different files, and this difficulty escalates with many of the project's files.

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 does offer sufficient commenting; the variable naming is also very appropriate and relevant to its purpose. The classes, methods, and functions were briefly described highlighting the purpose of those items in a clear and concise manner. However, the commenting is redundant and over-commented. The comments should explain why i.e. the reasoning behind the code and why was something done a certain way based on requirements. Additionally, a few of the comments included typos and lacked appropriate grammar and punctuation which decreased code legibility. Lastly, a few print lines were commented which can be said were used for debugging. The commented-out code can be confusing to a reader who has no context and therefore an improvement could be using Git for version control to track code changes and pushing the latest code without the debugging commented codelines.

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 followed appropriate indentation but the newline formatting deployment was inconsistent; An example of this is when initializing variables - a new line was placed between each initialized variable but when variables were initialized on the heap new line was not placed between the variables on the heap. An improvement could be being more consistent with the deployment, ideally choosing no new lines for initializing variables and can be placed in one code block as they all serve one purpose which is initializing variables. The parentheses placement was slightly inconsistent as for some functions the leading parentheses were placed on a newline and for some, they were not. Therefore an improvement could be having

consistent parentheses. Sensible white spaces were placed to improve readability such as between function parameters, for loop parameters, and initializing variables.

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)

Overall Gameplay:

Yes, the gameplay consistently delivers a remarkable level of smoothness, showcasing its resilience even during parameter experimentation such as changing the delay constant.

Collision Detection:

The precision of collision detection logic is evident in the execution of the suicide feature. By increasing the delay constant I was able to see how the suicide function worked and saw that it follows the logic of if the head position is on another body segment it ends the game

Food Consumption:

When the snake makes contact with either regular or special food items, the system exhibits swift responsiveness by promptly updating scores. Upon closer observation during a slowed-down sequence, it became apparent that the moment the snake's head aligns with the position of the food, it instantly disappears, triggering an immediate update of scores. This seamless interaction holds true for both regular and special food.

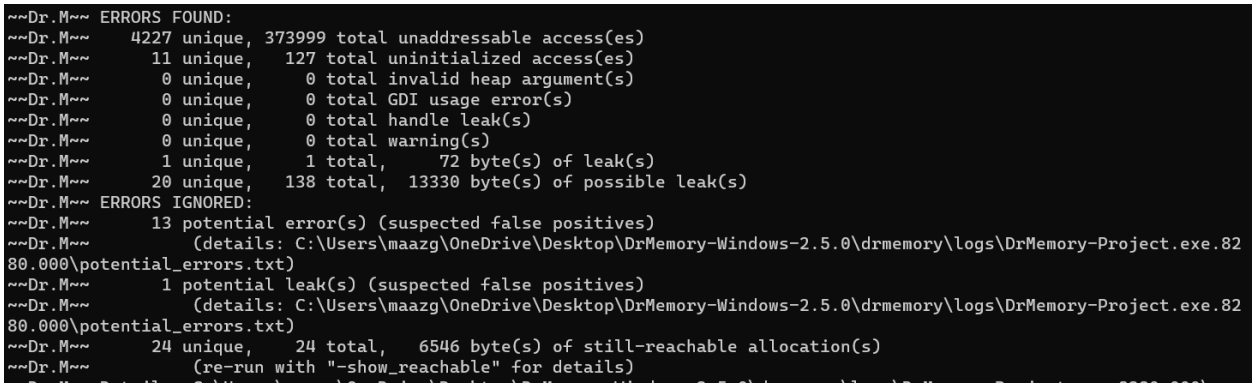
User Input Responsiveness:

The level of responsiveness in the inputs and the broader UX significantly enhances the gameplay. The WASD inputs are pretty responsive however when I try to make sharp turns in the game it doesn't always respond. This is mainly due to the delay constant as information might not be handled that fast. By decreasing the delay, I was able to make sharper turns more often.

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.

No, there is no memory leak in the Snake Game. As well as there are no invalid heap arguments meaning the group is very cautious about how they are controlling their memory access. By looking at each individual file of the project, every time the group was allocating memory to the heap, they made sure they were either adding a destructor to make it automatic or added the deallocation later in the file. A great example is the project.cpp file where the group allocated memory, *myGM = new GameMechs(30,14); myFood = new Food(myGM); myPlayer new Player(myGM, myFood);* but later deallocated it in the clean up;*delete myGM;*

delete myPlayer; delete myFood;



Ignore the 72 bytes, that is acceptable

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 you.

The collaboration was fun as it resembles a workplace environment. Having different people working on different things but then integrating each other's work to create one is a "soft skill" that can be used anywhere.