GROUP REVIEWING: Zijie Zhai and Ziyi Chen

Part 1: OOD Quality

1. While looking at the header files of the player, GameMechs, ObjPos and ObjPosArrayList, a high degree of interpretation is conveyed. The code includes good naming convention of methods and parameters going in and out of the header files. As well the code uses good comments to further add to the legibility and ease of use of the header files. One con this code has with header files is the implementation of the snake food without having its own class. Although the food does not need to have its own class and header file to work in the game, it makes it significantly easier to interpret if it was implemented in its own class.

2. In the main project.cpp file all objects are instantiated within the first 20 lines of code. These objects have a good naming convention and are easy to understand what they are referring to, (myfoodpos ,myplayerpos). When checking if the food object is touching the player, a very intuitive method is used. An object list of the player is input and a for loop checks with each item in the list whether the new food position is in the same X, Y as any of the player. Another pro of this code is the low usage of the Objpos, in my own code we created a temporary objpos multiple times to check different things within the player and the food. In this code only one or two objpos's are created in the draw screen. This makes it much easier to interpret as a low number of variable names are used.

3.

Procedural	OOD
CON: Functions do not access specifiers, meaning that to minimize the risk of accessing variables you don't intend to modify, more obscure naming conventions must be used (ex. num vs __num). This works but it is tedious.	PRO: Allows for better security of class elements through the use of private and protected scope, minimizing the risk of modifying variables accidentally.
PRO: All of the program's functions are contained within one file. This makes it easier to edit, read and compile.	CON: Multiple files must be constantly used and modified separated to eventually be compiled and executed together. Although VS Code's split screen mode helps with this, it can quickly get messy/confusing for large scale projects.
CON: Harder to increase functionality of the program. New tasks added to the program may involve changing and adding many aspects of the code instead of referencing already created methods or functions.	PRO: Allows for modulization of the program. If increased scope or functions need to be added to one's program, an OOD program may be able to reference previously created methods to create new functionality. This could be seen in the food class created for the project as it references the objpos class and the object array class in its creation, making it much easier to implement in the code.

Part 2: Code Quality

1. The code is very concise and organized, with large numbers of comments describing the functionality of each block. Also, the comments feel modular, allowing viewers to follow the general program process by reading them in order down the program file. The comments are also concise and to the point, not needlessly overexplaining simple functionalities. The functionality of each individual file outside of the main project file is also thoroughgoingly described in much the same way, meaning that even someone with very minimal understanding of programming would be able to understand almost exactly how each part of the program executes.

2. Proper indentation is used within all functions throughout each class, making loops and if statements very easy to read and quickly understand. Also, variables and unrelated lines within both the main program and individual classes are separated by spaces, which also vastly improves the code's readability. If anything, maybe a few too many spaces are used at times, separating parts of loops that would have already been spaced from lines containing only curly brackets, with additional lines being used to space them out further feeling unnecessary.

Part 3: Quick Functional Evaluation

1. Overall, the gameplay experience is very smooth and bug free. Even after playing until a relatively large snake size of 35, where a more unoptimized program make start experiencing issues, the input to response time was extremely fast and I was able to continue playing until even higher scores. The only small issue that I noticed was the delayed loading of the gameboard on each program loop, causing a reoccurring, extremely visible disappearance of the board. However, this did not affect the actual function of the game and was most likely due to the relatively large delay time of 0.1 seconds for each program loop coupled with the large default board dimensions of 36x18. I would argue this delay time is a good thing, however, as although it makes the gameplay feel slow it also works to prevent issues during later stages of play due to the program executing too quickly, which I think contributed to the lack of additional errors.

2. The snake game causes 0 bytes of memory leak. This means that this group has properly implemented destructors into every class that created elements on the heap; most importantly for the array list within the ArrayList.cpp file as well as proper delete statement implementation at the end of the main program for both the GM and Player instances declared on the heap.

Part 4: Your Own Collaboration Experience

Overall, we found this project to be a fairly positive experience. Although we have done coding assignments in partners previously in first year, this project was of a much larger scale and required more coordination. we found it very helpful to have someone to talk through my design process with, and we feel this helped us develop a deeper understanding of object-oriented design than we would have had working on it alone. The main issues we experienced were with GitHub merge, not to fault of the teaching team, which caused some issues even when working in different files exclusively. This led to a desire to work on the project at different times to avoid errors, which directly opposed the style the

project was meant to be taken on under. A simple solution we found was just to clone the file again when this happened, but that was very tedious. We think a quick lesson on the tools within GitHub, and how to use merging, cloning and branches would have helped smooth out the tedious experience our group had. As well as giving us good skills for further group projects implemented with GitHub.