Team Members: Kiana Wilding and Eloise Nguyen
Peers Evaluated: Humza and Mustafa

Part I: OOD Quality

1. Most of the header files are easy to interpret in terms of their contributions and behaviours in the program. However, the objPos.h file is not as easy to interpret due to its repetitive naming of objPos functions. Despite that, there is nothing we can do about the function names as objPos() is the default constructor and objPos(objPOs &o) is a copy constructor to take reference to objPos with 'o' as a parameter. Other than objPos.h, all other header files have distinguishable function names that are often self-explanatory.

2. The objects in the main loop are interpretable due to their self-explanatory names. The call sequence of the objects also follows a logical order of initializing variables, running the game, and then cleaning up. Having the game logic in a while loop also makes sense, as the actual game is the only part that needs repeating. The steps within the while loop are also arranged logically, by getting the user's input first before processing the input and running the logic to draw the game board, the game has all the appropriate parameters and inputs to proceed to the next feature.

3. Pros:

- Less global variables (less static storage) in the main program. This can improve code maintainability and reduce the risk of stack overflow with less static stack usage.
- High level modulation of game's features and intuitive object interaction. This promotes a clearer and more understandable structure of the program so that independent features are separated into their own classes and objects.
- Simpler heap allocation and deallocation, since the use of classes simplify and organize allocation in constructors, and deallocation in destructors.

Cons:

- Requires referencing of different classes to access their member functions. This can lead to more complex code, especially if there are dependencies between classes.
- Requires more discipline in heap management to ensure all allocated heaps are deallocated. Failure to do so can result in memory leaks.
- Limited overview of the entirety of the program as there are multiple files to reference, making it more difficult to change a variable as that requires alterations across multiple files.

Part II: Code Quality

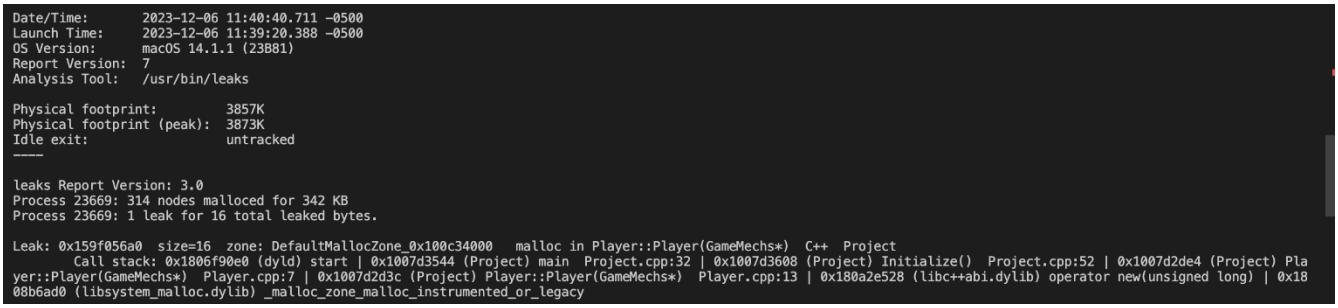
1. The code does offer comments however more can be included for further clarification. For example, in GameMechs.cpp the default constructor as well as the board constructor has the same board initialization, and it is unclear why they decided to do so. To fix this, we would remove the 1st instance of the gameboard initialization loop and only include the memory allocation for the board on the heap in the default constructor and keep the 2nd instance of the gameboard initialization loop in the gameboard constructor function.

2. For most of the program, the code follows good indentations and newline formatting for better readability. However in the Project.cpp file, the DrawScreen function is a bit harder to follow due to the nested for loops and if statements. In order to make this function slightly more concise, we would combine the snake food placement and game board placement into one loop and print them in a more logical order, with the game board being printed first, then the food being initialized second.

Part III: Quick Functional Evaluation

1. Yes, the Snake Game offers smooth bug-free playing experience particularly on MacOS. However, when using windows there is more glitches and bugs in comparison likely due to the different operating system. In addition, the while loop that loops through the game logic depends on the gameState's ExitFlag and LoseFlag statuses as its conditions. The use of the AND operator would nullify the escape key's function as the game can only exit if the ExitFlag and LoseFlag are false. This means that the user can't escape the game if they only trigger the ExitFlag, they must also lose. One way to fix this is to simply employ the ExitFlag as the only while loop's condition.

2. The snake game does cause 1 byte of memory leakage after running a DrMemory check (mac version). This likely due to the function Player::Player(GameMechs *thisGMRef) in the player.cpp file, due to having initialized two instances on the heap, and only deallocated one of them. Since only one instance is deallocated, the other one is not managed causing memory leak and could potentially lead to the program crashing.



Team Members: Kiana Wilding and Eloise Nguyen
Peers Evaluated: Humza and Mustafa

Part IV: Your Own Collaboration Experience

1. The project experience in a group of 2 went rather smoothly, however, we could have improved our communication and code implementation if we had collaborated on the first iteration. Usage of different OS also resulted in additional program hindrance as there would be clashing makefiles and MacUILib.h. Git pull and push between multiple devices can also produce clashes and duplicates of data if not implemented carefully, which can cause loss or errors of data and further frustrations. Troubleshooting errors were much more efficient with a partner thanks to their second opinion. The workload is also much more manageable as half of the program is your partner's responsibility.