COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Jiayi Yang and Xiang Li

Team Members Evaluated Ryan and Mani

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions.

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.

- **Positive features:**

  Proper encapsulation: The use of getters and setters, as well as the private visibility of member variables In Play, GameMechs, Food and objPosArrayList classes adheres well to the encapsulation principle.

  Clear responsibilities: The proper usage of pointers and references from each class for dynamic allocation of objects indicates high cohesion. It also increases the responsibility for memory management.

- **Negative features:**
  High Coupling: There's significant coupling between classes. Player directly calls methods on GameMechs class and Food class, and Food requires knowledge of the Player's positions to generate food. This tight coupling makes the system more rigid and less modular.

  Potential for Circular Dependencies: The Food class requires a reference to objPosArrayList in void generateFood（objPosArrayList* arrayListRef） method, which is also used by Player. This could lead to circular dependencies if not managed carefully.

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.

- **Positive features:**

  Clear Player-Game Interaction: The 'Player' class effectively interacts with the 'GameMechs' and 'Food' classes in the main loop. In the 'Player' class, 'updatePlayerDir' method captures user input and modifies the player's direction based on the input and the game rules.

  GameMechs and Food Coordination: The 'Player' class communicates with the 'GameMechs' and 'Food' objects seamlessly to update the game state. Having the 'Food' as a class separately indicating high cohesion.

- **Negative features:**

  Method Implementation: The methods suggested in iteration 3 for checkFoodConsumption(), increasePlayerLength() are not implemented in the project. Instead, movePlayer() method that inside the Player class contains the logic for them. By following encapsulation principle, each object encapsulates data and behavior can help leading to a more flexible structure. By following polymorphism principle, seperate different funtionalities in different methods can enables code reusement and make maintenance easier. Therefore, using the suggested methods rather than combine them would be better for interaction between functions when implementing in main project and benifit the high-level flow of the game.

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: Groups related data and behaviors into classes, reducing global state and side effects. Makes the system more extensible, allowing new functionality to be added with minimal impact on existing code which benefit the workflow of iterations.

  Modularity: Enhances the modularity of the application, making it easier to understand and maintain separate components like player objects can be separately modified inside class.

  Reusability: Allows for code reuse through inheritance and polymorphism, which can lead to less redundant code within the main project for Runlogic() approach.

  Abstraction: Provides abstraction mechanisms to hide complex implementation details and expose only necessary interfaces in the main project.

- **Cons:**

  Complexity: Increase additional complexity with the need to understand the connection between classes, objects, inheritance.

  Overhead: May add performance overhead due to features like dynamic memory management.

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 exhibits a generally commendable level of comments, aiding in understanding the functionality efficiently. Positive aspects include descriptive comments for class declarations and method headers providing insights into their roles. Most of the variables are provide with clear comments contributing to self-documenting code. The only thing could be improved I'd say is deleting the comments which was left for the programmer as a to do list after all blocks are done if there would be other programmer going through the code again.

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 adheres to good indentation utilizing consistent spacing and newlines for improved readability. Positive aspects include well aligned blocks, making the code structure clear and easy to follow. Variable declarations, method definitions, and control flow structures are appropriately indented, contributing to code organization. One thing to mention is when defining a function, having the opening curly brace with the closing curly brace in the same columns would improve the readability. Additionally, when printing out the debug messages, the player position should be printed in the same line as the text "Player Position:" to improve the readability.

Part III: Quick Functional Evaluation

1. **[8 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 a technical user feedback)

   The Snake Game demonstrates a commendable level of stability and a bug-free playing experience. The implemented features, such as player movement, food generation, and score tracking, operate seamlessly, contributing to a good user experience.

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 leak.

```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       4 unique,     5 total uninitialized access(es)
~~Dr.M~~       1 unique,    62 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total GDI usage error(s)
~~Dr.M~~       0 unique,     0 total handle leak(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
```

By running the Dr.memory, there is no memory leaks with the program.

This game has a great memory control as it allocates memory on the heap and deallocate at the correct part. For every class that needed to allocate memory, the destructors are all included inside the class to deallocate memory. For example, the screenshot below indicates the process of allocating memory in objArrayList class and how aList variable is deleted in the corresponding destructor to prevent memory leakage.

```
// Constructor for objPosArrayList
objPosArrayList::objPosArrayList() {
    arrayCapacity = ARRAY_MAX_CAP;  // Set the capacity of the array to a predefined maximum
    listSize = 0;                   // Initialize the current size of the list to 0
    aList = new objPos[arrayCapacity]; // Dynamically allocate memory for 'arrayCapacity' number of objPos objects
}

// Destructor for objPosArrayList
objPosArrayList::~objPosArrayList() {
    delete[] aList;  // Delete the dynamically allocated array to prevent memory leaks
}
```

Also in the main project file, memory allocattion happens when initializing gameboard size, player position and food position. After that, all dynamically allocated memories are deallocated inside the clear up function explicitly using delete and avoid overall memory leaks.

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.

   We are a two-person-team. The overall coding experience was fine. I think this provide an exposure of what would happen when we as working as an engineering team in the future. The time span of the project was sufficient for the collaboration to get the work done.