COMPENG: 2SH4 Project – Peer Evaluation

Your Team Members **Matthew Rozema, Noah Singh**

Team Members Evaluated **Arya Pirashody, Peace Peace**

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.

Looking at the header files of each object that this group involved in their program it was very easy to interpret the behaviours of objects involved in the program. For instance the header file names were kept consistent with the names of the CPP files and the purpose of the file. For example, the header file "Food.h" clearly can be interpreted as interacting with the food generation portion of the program as the name clearly states. This can be seen with every header file of the objects within the program, making the code relatively easier to understand at a quick first glance.

A positive about their sensible code modularization is that it allows for the ability to update individual components of the code along the way if there were any errors by easily being able to locate and fix the error without having to affect the rest of the code. As well, by organising their code in such a way it will be beneficial if in the future they wanted to attempt the bonus of the project since it would be straightforward as to what object does and what should be modified. All in all, the use of their header files for each object allowed for proper OOD code modularization which helps any viewer of the code understand the purpose's with a quick observation.

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.

From examining the main logic within the main program loop, it is relatively easy to interpret how the objects interacted with one another. For example inside the "RunLogic" function the sequential calls of "updatePlayerDir()" and "movePlayer();" provides a clear and logical flow of how the interactions of each loop in the main program begin, by updating and moving the player.

Inside the “DrawScreen(void);” function, the code is written in a concise way that details the process in how things are drawn onto the terminal. Specifically, the ‘thisGMRef’ pointer to update the score shows how an interaction between objects occurs through each loop iteration. As well within the initialization routine, the creation of pointers provides transparency on how objects such as ‘Food’ interact with food related operations (generation). As well, the use of the ‘thisGMRef’ pointer enables a reader to quickly discern how objects will interact with functions that rely on the game mechanics. In summary, the way in which this code was written, allows for easy interpretation of how each object created works with each other toward the shared goal of executing this modular C++ program.

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 C++ OOD Approach

- Readability through the uses of classes and methods which allows for easier comprehension of the code.
- Modularity through the encapsulation of objects such as Food, Player, and GameMechs. This allows for the code to stay more organized while working on it.
- Flexibility by the use of dynamic memory allocation which allows the programmer to manage lifetimes of certain objects.

Cons of C++ OOD Approach

- Complexity as it is not as intuitive as procedural programming.

Pros of C Procedural Design Approach

- Simplicity as it is a lot more intuitive to grasp.

Cons of C Procedural Design Approach

- Scalability as when the project starts to get more complex it becomes harder to maintain.
- Difficult to keep organized since there are no classes or objects to aid in that area.
- Limited abstraction which can lead to much more complex code as the project grows.

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.

While their code provided a clear and sufficient self-documenting coding style, there were certain sections where having more comments could have benefited a first time viewer of the code. For instance in the main program loop or near the self collision check inside the Player.cpp file, the addition of extra comments could have provided further insight into the functionality and rationale. However, in places where sufficient comments were included, it really helped in understanding the approach they took in tackling different aspects of this project.

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.

Yes, the code does follow good indentation practices and allows for easy readability. Their use of sensible whitespaces and well placed newlines helps to contribute to an uncluttered piece of code. Additionally the deletion of unnecessary lines of code adds to the overall conciseness. These practices that they used within this project made the code very approachable for viewers and helps in the comprehension of it.

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)

Controlling the snake feels smooth and there does not seem to be any input delay. However there is a small issue found when the snake collides with the food object. After the collision, the snake stops moving briefly, then the food generates to a different position, after which the snake's movement continues. This however causes the player to briefly lose momentum while controlling the snake.

This issue is likely due to how the team chose to increase the snake length. Specifically, in the increaseplayerlength player class method that gets called after a collision with the food, the method adds a new player object at the head of the snake with the same objPos x and y values as the previous head object. This gives the impression of the snake briefly standing still as the movePlayer method needs to be called once to move the added body object to the front of the snake, and then another time to properly begin movement. This also meant that in the drawScreen function in Project.cpp, the team had to add extra safe guards to not print out both head objects on the same coordinate, which would mess up the border. A potential fix would be to just not remove the tail when moving the snake, as since their program does not attempt the bonus special characters feature, they will always be incrementing the snake size by one when colliding with food. This reduces the need for the movePlayer method to be called twice to properly move the snake.

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.

Fortunately, the snake game does not cause memory leak, as the development team properly implemented destructors for the objectPosArray and Player classes which both created variables on the heap in their constructors, using list delete (delete[]) for objectPosArray to free every element in the list at once. They also freed variables created on the heap in the initialize function as part of the clean-up routine in the correct order, to not lose access to specific references. Specifically, in initialize the GameMechs object was created, which was used in the Food object's construction, and both were then used in the Player object's construction. Due to this, when freeing the variables on the heap, they freed the Player object, then Food object, then GameMechs object.

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.

Throughout our first collaborated software development we were able to successfully work together to finish our project. Our collaborated work style really benefited us during the first two iterations, since we could evenly split our work, and then had one person bridge them together. Additionally, since one's work required the other, we were more motivated to meet deadlines and get everything done on time.

However, during iteration 3, we found it was hard to split up the work since the entire iteration had to be built up sequentially. This made it easier if we just had one person do iteration 3 so that their overall vision and design approach stayed consistent, while the other layed out groundwork for the bonus.