COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Natalia Jara and Sofia Perme

Team Members Evaluated Ayush and Kirit

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, I can easily interpret most of the behaviours of the objects in the program. Positively, the names of the fields and methods seem to provide a general description of its purpose. For example, in the food class, a public method called incrementCollectedFood() is defined. The name of this method gives me the idea that it keeps track of the food being collected by the player.

As a critique, one thing I noticed in the main program is that the GetInput() routine uses procedural programming, instead of referring to the input collection from GameMechs() class. This defeats the purpose of having an input getter in the first place.

Another concern of ours was the CleanUp() routine in the project file. This function uses some elements of procedural coding by both checking for the exit and loseFlags and printing these messages when this routine was only supposed to be used to clean up and exit the program. While this is correct, it does not allow for the full use of encapsulation that helps make OOD so useful. An additional method could have been created in the GameMechs() class and then called in the main program to reflect these end game actions, better implementing OOD design.

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.

While looking at the main code, it is very easily readable and distinguishable. The main logic is simple and easy to read with descriptive variable names, while also adding a few comments where the variable names are not descriptive enough to compensate. For example, in the DrawScreen() routine, they specified which blocks of code were responsible for printing each element/object. They use descriptive names for the references to the Food(), Player() and GameMechs() classes which helps us easily distinguish which objects are interacting with each other. They mostly used the default names of myPlayer and myGM to represent the Player() and GameMechs() classes, however they decided to use candy as the variable name for their food class. This is a good decision since it is still easily recognisable as a food item, but also to their advanced features addition of special food objects, cherry and watermelon. In addition to good variable names, the output messages they printed such as the exit,

losing and winning message were descriptive enough that they almost acted as comments and explained what was happening in the code.

In the main program, they added a GetInput() routine, which although defeats some of the purpose of the OOD design also makes the main logic skeleton look incredibly like that of the PPAs, which helps with readability due to our familiarity with the PPAs. The procedural element of this code makes it easy to understand from the main logic due to procedural code being less complex and the having more intuitive logic.

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:
   - **Object Interaction and Reusability:** In the project, creating multiple classes with their own public methods that contain the appropriate features and functions from PPA3 allows for efficient interaction of objects and various components.
   - **Organization:** In the project, multiple classes interacting with each other and simply being called in the appropriate main logic routines allows for more overall organization of the code; the main program code is cleaner, improving readability.
   - **Encapsulation:** In the project, private information can easily be hidden from the public to prevent unexpected program access. Only certain objects and methods can access this private information if specified, improving the security of the code.

   Cons:
   - **Complexity:** In the project, C++ requires you to be very disciplined – must have a proper planned out program to begin, must keep track of object interactions, strict code organization, and other strict syntactical C++ rules.
   - **Tracking Program Flow:** When an error presents itself in a C++ program, debugging can be a challenge. It may be difficult to tell whether the bug is coming from a class method, main program, etc. In C, it is easier to track the program flow as the procedural design approach is fully implemented in the main program.

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.

After looking through each class and the main program, the code offers sufficient commenting for the most part. There is a comment above every code block that describes its purpose, helping the user understand the code in greater detail. For example, in the food class, their generateFood() method is very descriptive. In comments above corresponding code sections, the random food generation process is described in words – generate random coordinates, checking if the coordinates overlap with any existing ones, etc. Some more comments describing the purpose of initialized variables in certain functions would be helpful, as some variables were not descriptive enough in their names.

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.

Overall, the code is very readable and follows proper indentation, newline formatting and sensible white paces. I have a few minor criticisms, such as the placement of curly brackets will follow two different conventions: one which they are placed in a new line after the line of code and the other at the end of the line of code. Although this is a minor thing, I would just format the document to choose one convention and stick with it. Another minor thing is that they could remove a couple of the empty white space lines, this is something that happens very infrequently and at a small scale. For example, in player.cpp lines 180-183, I would delete a line or two of the empty lines to just reduce the number of lines and make the readability slightly better. These are incredibly small points and even without any of these easy changes the code is incredibly readable.

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 game playing experience is very smooth. The snake/player object itself is running at a good speed and there are no lags of any kind. The game also always generates 5 new objects, two of which are special objects. The game has great messages for user playability- especially regarding the special objects and what they do. The self-collision works smoothly and the game playing experience is a very good one from the technical side.

However, the game itself is a little boring due to some developer decisions regarding the way that they decided to implement the advanced feature. Instead of regenerating all the elements when the first player-food collision happens the user must get all 5 of the food items to regenerate the next 5. The generation of the number of special/ordinary food items is also always the same and this makes the game very boring and predictable. In conclusion, the game itself runs bug free and fulfills all the necessary components of the snake game, however, the developer's implementation of the advanced feature made the game boring to play.

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.

After running drmemory for the Snake Game, the memory profiling report indicated that there is no memory leak. When examining the main project file, all objects that were created on the heap, as initialized in the Initialize() routine, were also deallocated in the CleanUp() routine using the delete keyword. Taking a closer look at each of the class header files, three private fields were defined on the heap, one from each class – Food class, GameMechs class, and objPosArrayList class. Taking a closer look at the files of each of the classes, the destructors are filled in appropriately with a line deleting each of the private fields that were originally defined on the heap. For these reasons, the Snake Game does not cause memory leak.

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.

Our first collaborated software development experience presented us with a lot of challenges and learning opportunities. We found that it was a great way to introduce us to git and help us adapt to this online platform that is widely used in the industry today. One thing we found difficult was coordinating changes with each other when working separately. Even though we had designated features and classes to work on, sometimes our changes overlapped with each other's classes. We often ran into merge conflicts when pushing our codes, which was quite irritating at times. For this reason, we found ourselves communicating a lot through text to let each other know of changes made. When working with each other side by side, it was way easier to simultaneously work on the code, while communicating program updates.