COMPENG 2SH4 Project – Peer Evaluation

Team Members: Prawin Premachandran and Aun Maqsood

Team Members Evaluated: Leandra and Aditi

Part I: OOD Quality

1. Header files are important to understand the OOD principles that were used to tackle the program, as well as the interconnected nature of these classes as they interact with each other. The team clearly laid out all of the methods and data members in chronological order, such that an equivalent UML diagram could be made without much effort. For example, the team's food class included instances of objects of different classes under the private scope, indicative of composition and functional class implementations being used. The members could be mapped to their classes (objPos and gameMechs, which are all functional classes) to then inspect their header files and definitions. The constructors, destructors, getters and setters were all laid out in an easy-to-follow manner. A future suggestion might be to include more comments for the data members which were created as instances of other classes, about their usage in the class or the object.
2. Inspecting the program, there was little to criticize. The program loop is self-explanatory as it is, with each method in the loop describing the procedural steps needed for the program to function. The objects which are declared were semantically meaningful and therefore accessible from an outside perspective. The initialize routine described in chronological order, the objects that needed to be initialized, and how they were initialized using the respective methods that belonged to those instances. These initializations were so intuitive that minimal comments were needed (for example, in order to generate food, the method generateFood was used via arrow notation, because a global food pointer to the food object was declared on the heap). The logic of the other sub-routines could easily be discerned by the methods describing actions applied to each object taking place. These actions could be traced to the respective class implementations to gauge the underlying mechanics of each aspect of the game. Such modularization and abstraction was easy to follow and also allowed a foreign developer to conceptually understand the design. The draw routine was relatively harder to grasp, due to the complexity of the triple nested for loop. Still, the meaningful variable names and appropriate use of methods made it easier to interpret. A future suggestion would be to use semantically meaningful names for the local temp loop variables (rows for the outer, columns for the 2nd inner, maybe something relating to the snake body for the 3rd

inner). The loop delay and cleanup subroutines are obvious to a developer familiar with the syntax of C++, knowing that deallocation is needed to prevent memory leakage.

3. The pros and cons of C++ OOD instead of C procedural programming (described below).

Pros:

- C++ OOD approach is that is promotes modularity and hence makes it easier to organize and follow code. This is because each class can be referenced in other parts of the code.
- The C++ OOD approach promotes code reusability. For example, in our project, a specific functionality such as updatePlayerDir() from the player class can be called several times within the program without the need for duplicating code. Additionally, group members who wanted to do the bonus could use inheritance to extend their class functionalities without copying code.
- C++ OOD prevents repetitive code. If we needed to do this project with the procedural approach, we would not be able to use classes, which would result in code redundancy. In general, C++ OOD is much more efficient than procedural programming when doing big projects that require several lines of code.
- The OOD principles promote abstraction of code, which means that a developer can interact with one aspect of the code without understanding the underlying implementation and specifications. Only an intuitive understanding of its physical meaning is needed (I.e one doesn't need to understand the underlying implementation of the game mechanics class to use it in the player class implementation).

Cons:

- May be too complicated when working on simpler projects such as in PPA 1 and PPA 2. Here, it would be much easier to use C-procedural programming approach.
- May be too difficult for inexperienced/beginner programs to learn C++ OOD approach
- Memory management challenges can occur through destructors in C++ if the heap variables are not carefully tracked and deallocated. As discussed in class, memory leaks on a bigger scale can result in worse computer performance over time, and it can also cause programs to occasionally crash.
- Parroting the last point, efficient memory management can be easily done through procedural programming, as reusing code is a much more conscious decision than when using an object. More low-level tasks would benefit from such a style.

Part II: Code Quality

1. . The comments in the underlying class implementations of the program were laid out nicely. Any functional classes were noted by mentioning that composition was used (I.e player composed of objPosArrayList as a comment). The finite state machine and movement logic was adequately laid out and easy to understand. There was no over-commenting to add redundancy for the developer in these implementations. The

project file with the program loop was also fairly well-documented for the most part, with no redundant comments. A shortcoming that was observed was the lack of documentation for the draw routine, which served as the most complicated routine to understand conceptually. Comments to describe the loops in more detail, as well as choices for why certain actions were used (I.e the significance of the Boolean drawn variable) would facilitate a greater understanding of design choices in this routine.

2. Yes, the code is formatted well with good indentations under each function (e.g. main, initialize, draw screen functions). The code is well spaced, which enabled our team to easily follow along and interpret their code. We didn't observe any shortcomings.

Part III: Quick Functional Evaluation

1. For the most part, the group's Snake Game offers a smooth experience, and both exits (suicide and forced exit) work successfully based on multiple trials. Their snake speed is set at a good speed/difficulty, as it allows the user to have an easy start to the game, and then the game starts to get more challenging as the user's score increases. Perhaps, if we were to change anything, we would make the board size bigger (e.g. 30 x 15) instead of their current board size of 20 x 10 to make the game last longer. Since it was observed that after reaching a score of 10, the snake would suicide quite often after going through the vertical wraparound function because the head would easily collide the tail. Also, after reaching a score above 10, it would be difficult to control the snake because it is hard to determine where the snake head is (after a wraparound). This can be fixed with a bigger board size.
2. No memory leaks. It is observed that the group allocated variables on the heap including Player, GameMechs and Food. And they prevented any memory leaks by deallocating the heap memory by deleting it in their CleanUp() function.

Part IV: Your Own Collaboration Experience (Ungraded)

Overall, it was a fun experience for both of us and we enjoyed working together on our first collaborative project for 2SH4. Initially, we started off the project by reviewing topics 12 and 13 on OOD and array lists to ensure that both of us were caught up with the class material. Next, we split up the work accordingly, where Aun worked on iteration 1a) and 2a) while Prawin worked on iterations 1b) and 2b). Small issues were initially encountered such as the display screen bugging and the wraparound feature. However, we helped each other and discussed consistently through video calls and in-person meetings. Eventually, we met together again to collaboratively work on iteration 3 and resolved any issues we faced. Reflecting on the process, we are satisfied with our consistent efforts and collaboration as we were able to make a successful snake game.