COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members: Shajijan Narendran (narends), Alex Melnbardis(melnbaa)

Evaluated Team Name: drchen ./IsAwesome.exe

Team Members Evaluated: Erin Herzstein (herzstee), Lucy Schartner(schartnl)

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **25 marks**. Do not exceed 2 paragraphs per question.

Peer Code Review: OOD Quality

1. **[3 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.

For the most part, it was easy to understand the main program code. The initialization and allocation of memory for the all the objects used within the code is correct and concise. The draw routine makes very smart use of the Player and GameMechs objects to efficiently check that all the objects are printed in the correct location. One drawback of the code lies within the Run Logic function. Although it may be purely subjective, it seems unnecessary to include the exit key input separate from the rest of the update Player Direction functionality. Hypothetically, if you were to place the exit key input collection with the rest of the controls (W, A, S, D), then there would be less code that needs to be written. Overall, the main program is very clear and interpretable with one minor nitpick.

2. **[3 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.

C++ OOD Approach

Pros:

- Encapsulation: data and methods can be bundled into classes which prevents code from “spilling” into other areas
- Modularity: the use of classes allows for code reusability and makes it easier to add-on new features later, such as the bonus

Cons:

- Complexity: OOD leads to a small project such as this (in C) becoming big and complex to manage

C Procedural Design Approach

Pros:

- Simplicity: the procedural approach is better for small projects as it is more straightforward and requires less code; PPA3 is small enough to be readable in a single file

Cons:

- Poor scalability: as the project gets bigger, the lack of modularity makes it harder to add new features without affecting the current logic present
- Poor encapsulation: the absence of classes also results in data and functions becoming more separated, making it difficult to hide internal states and enforce access control

Peer Code Review: Code Quality

1. **[3 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 includes many comments in each file which help to understand the purpose of listed functions and code blocks. In places where comments are not used, descriptive variable names are used to help create a self-documenting coding style. Overall, other than in some areas where code *implementation* could be improved, there are not obvious shortcomings present in code *quality*.

2. **[3 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.

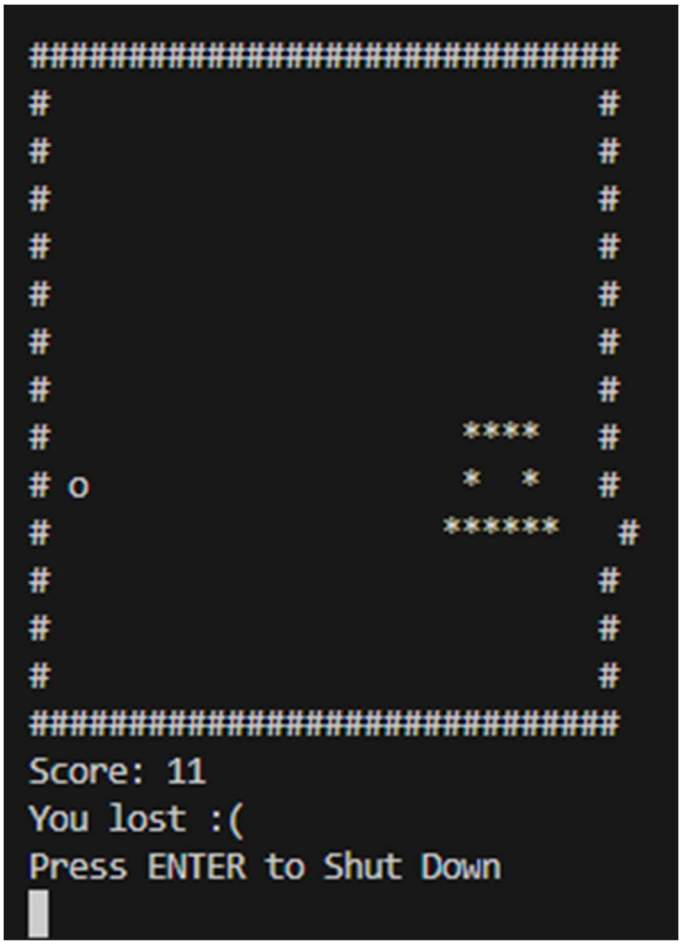
The code follows good indentation for code blocks and use of white spaces, as well as an acceptable use of newline of formatting. However, there are several areas that can be improved. Some shortcomings discovered include unnecessary white space at the end of code blocks (e.g. before the end of brackets), comments being sandwiched between statements (in-line with the code) and/or inconsistently placed and spaced, and code blocks sometimes containing too many statements put together at once without any newlines or white space.

To improve these issues, white space and/or newlines not helping to improve readability should be removed, comments should be given sensible space above and below them and be placed beside or on top of the code they are describing, and group together code statements similar only by their function/purpose (e.g. separate memory allocation from unrelated continuous assignment statements).

Peer Code Review: Quick Functional Evaluation

1. **[3 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 technical user feedback)

For the most part, the game runs smoothly, but there are a few bugs. After the snake death condition is initiated, there is a bug in the draw screen method. On the row in which the head intersects the tail of the snake, the walls of the grid are push one space to the right. This is because an extra snake character which is not part of the objPosArrayList (*) is incorrectly drawn on that same row. Our group ran into the same issue. This bug can be fixed by checking your itemPrinted variable as a parameter within each of the for loops.



The snake was going downward into its own tail. A random character is drawn on the tail and the hashmark is pushed one space to the right.

2. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.

The game does not cause any memory leakage. Pointers are initialized on the heap and deleted properly; all the necessary constructor and destructors functions appear to be implemented correctly.

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

We don't believe that the compound object design of the objPos class is sensible. Firstly, placing a struct in a class is somewhat redundant since a class already encapsulates the data and provides the necessary functionality. Additionally, since the struct is declared outside of the class implementation, its members int x and in y are considered public, which technically breaks the C++ OOD encapsulation principle; it is more common to keep data members private. Also, the use of the struct makes the code statement(s) required to access its members slightly longer and harder to read.

2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. You are expected to facilitate the discussion with UML diagram(s).

The object design could be improved by making the x and y position variables private within the class and creating getter and setter functions to easily modify the variables for each new object generated, whether it be a player or food object. This way of distributing the variables is much easier to read and allows easy access and alteration for the location variables. Considering how many times these variables are edited within the function of the program this new structure is much more efficient.

objPos
+ x: int + y: int + symbol: char

objPos();
objPos(int xPos, int yPos, char sym)
 	
~objPos();
objPos(const objPos &d);
objPos &operator=(const objPos &d);
 	
setObjPos(objPos o): void
setObjPos(int xPos, int yPos, char sym): void
 	
getSymbol() const: char
getSymbolIfPosEqual(const objPos *refPos) const: char
 	
isPosEqual(const objPos *refPos) const: bool
 	
//New Functions
 	
getxPos() const: int
getyPos() const: int
setxPos(int xPos): int
setyPos(int yPos): int