# COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Sameer Suleman, Aman Rajesh

Team Members Evaluated Jingyuan Deng, Haolin Du

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

## Part I: OOD Quality

1. This group has demonstrated a thorough understanding of OOD and Code Modularization in C++. All of their header's are displayed where they should be, provided with appropriate names and there is no conflict in file naming. It was very easy to interpret what each function and variable members of the method are supposed to do. What we really liked was that there were pointers to each object in the program clearly displayed on the file, making it easy to keep track of the programming flow and reducing the complexity of the code.

2. With respect to the group's main logic in the main program loop, we think that they did an excellent job! The code is clear to read, and it adheres to the modularity of C++. The only suggestion is that the "isSnackBody" function should be moved from the main function to a related class such as the Player class. It would make more sense to be here with all the other functions relating to the snake's body.

3. **C++ OOD Approach**
   Pros:
   - C++ is more catered to OOD, instead of having superstructs like in C, we have dedicated data types known as classes to group related functions. This made it really easy for us to create a class for food, the game mechanics, the object positions, and the array of object positions (aka our "snake").
   - C++ supports function overloading, making it simple for multiple functions to be produced in a class, the user does not have to make a bunch of random function names that have to be kept track of, this way we can process multiple inputs for our constructors to initialize variables. We utilized this feature in the snake project when we were trying to obtain the position (or coordinates) of our objects of interest such as food, snake head etc. We could pass in the reference to the object or the x/y coordinates of the object we were interested in to get the coordinates which made it really flexible for our program.
   - C++ also has private/public/protected scopes for variables making it difficult for one to access values outside of the necessary class that the variable is associated with. This reduces the chance of the user making a mistake and

referencing/mutating a variable by accident. In our project, because we divided aspects of the game such as the food and the snake body and initialized variables under private/public scopes, it made sure that we didn't spawn a segment of the snake body on accident for instance if we just so happened to use the same variable name.

Cons:

- C++ does not have built in memory management, it is usually up to the user to deallocate after each memory allocation, this can be hard to keep track of in larger programs

Ultimately, C++ is quite useful for us to complete the project. C++ makes it really easy to keep track of and respectively update the segments of our "snake" and its interactions such as the game over condition or when it eats food. Moreover, the variable privacy ensures that there is no accidental mutation of variables.

**C Procedural Design**

Pros:

- C is a very powerful language, so it was easy to complete PPA 3 without encountering any error on the language's part
- Very easy to transfer code between files so it made it easy to import structs for use in other files

Cons:

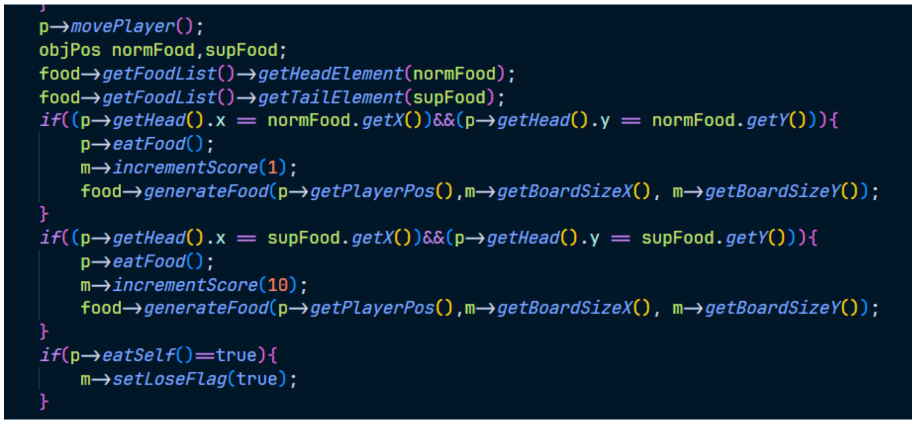
- Hard to define a public/private scope for structs in C so as they don't have dedicated built in functions to do this, so this may introduce error on the users part if they accidentally mutate a value that is supposed to be out of their scope
- C doesn't support function overloading, so when creating functions which have similar functions but accept different inputs, there have to be multiple different names, which can be hard to keep track of.

Ultimately, C is a strong language to complete PPA3 in, however, C++ is better fit for it, it would be easier to complete the assignment by creating classes for its different components, and keeping the code modular.

## Part II: Code Quality

1. For the most part, the code is well commented, mentioning how each part of the code functions. This can especially be seen in the Food.cpp file which is extensively commented, allowing me to easily follow what each part of the code is supposed to do. However, one specific part I noticed was that there were a lot of variable names with one word names such as “m”, “p”, “t”, etc. This can especially be seen in the main file where the gameMechs and Player object pointers were named m and p. I think it would be helpful to give these variables more detailed names so it is easier to remember what each variable does.

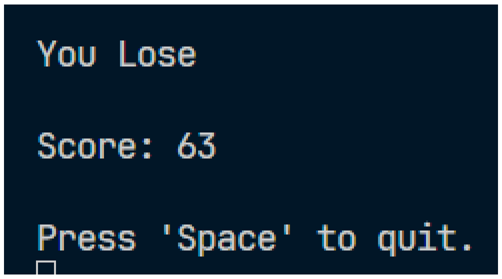
2. Yes, this code has good indentation, white space (for the most part), and newline formatting. The only minor suggestion I have is to have some whitespace before and after a conditional statement, as it can get jumbled to a big block of code without them (as seen below in the image). Other than this specific example the code was very well formatted throughout all the other files!



```
p->movePlayer();
objPos normFood,supFood;
food->getFoodList()->getHeadElement(normFood);
food->getFoodList()->getTailElement(supFood);
if((p->getHead().x == normFood.getX())&&(p->getHead().y == normFood.getY())){
    p->eatFood();
    m->incrementScore(1);
    food->generateFood(p->getPlayerPos(),m->getBoardSizeX(), m->getBoardSizeY());
}
if((p->getHead().x == supFood.getX())&&(p->getHead().y == supFood.getY())){
    p->eatFood();
    m->incrementScore(10);
    food->generateFood(p->getPlayerPos(),m->getBoardSizeX(), m->getBoardSizeY());
}
if(p->eatSelf()==true){
    m->setLoseFlag(true);
}
```

## Part III: Quick Functional Evaluation

1. The snake game is very well made, with a bug free and smooth experience while playing. It takes input correctly, grows/decreases the snake correctly, and implements the end game function and bonus correctly. However, I encountered a bug in the displaying of scores in the “You Lose” screen (shown below). Sometimes, the score would increase despite the game being ended. I believe this is because only the loseFlag was set to true when the snake collides with itself, meaning the game is actually running in the background despite not being displayed. If lined up correctly, it is possible for the snake to eat food after the game is over. To fix this, I recommend the team to set both exitFlag and loseFlag to true when the game is over. This completely stops the game from running then displays the score which would properly show the correct score everytime.

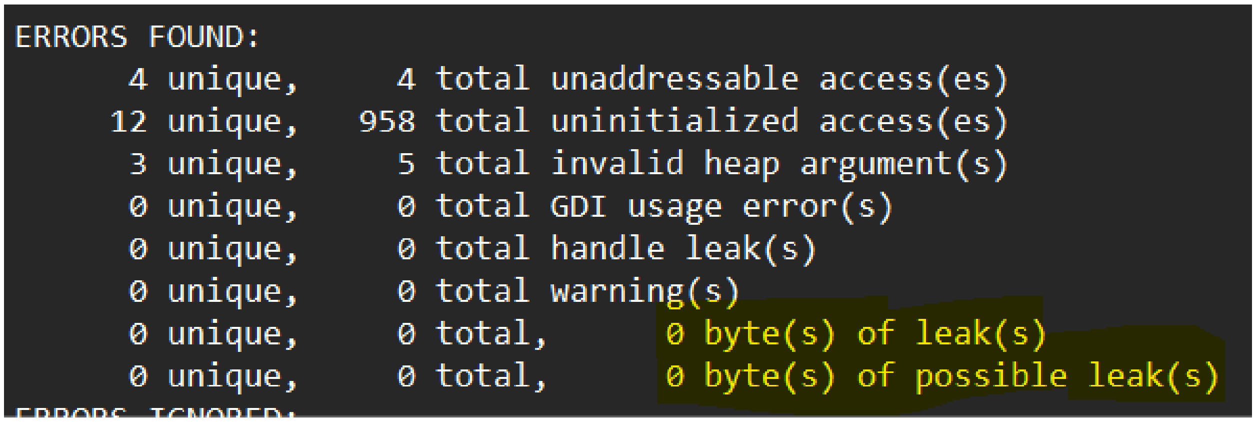


```
You Lose

Score: 63

Press 'Space' to quit.
```

2. The game has no memory leaks and no possible memory leaks. A screenshot of the memory profiling report is shown below, showing evidence of no leaks.



```
ERRORS FOUND:
      4 unique,     4 total unaddressable access(es)
     12 unique,   958 total uninitialized access(es)
      3 unique,     5 total invalid heap argument(s)
      0 unique,     0 total GDI usage error(s)
      0 unique,     0 total handle leak(s)
      0 unique,     0 total warning(s)
      0 unique,     0 total,      0 byte(s) of leak(s)
      0 unique,     0 total,      0 byte(s) of possible leak(s)
```

## Part IV: Own Collaboration Experience:

Our collaboration experience was truly insightful, we believe that we both learned a lot by undertaking this project. This was our first time creating a program with another user, so it was imperative that both of us followed proper programming practices. For instance, we both had to make sure our code had variable names that made sense,  that it was legible, and was also commented/organized well. Working together really enforced these practices on us, which is good because in the industry there are less room for mistakes like this, and there are higher stakes when working in the industry, so it's a good thing that we adapted early on. It was a little hard to get an understanding of the code when the other partner had completed their part, so it made it hard to get back into the flow of programming after each commit. I really did like how easy Github makes the code transferring process, all you have to really do to incorporate your partners changes is to pull from the repository, there's no need to copy paste anything or do anything which might make room for errors in the code! Overall, we found it to be a great experience because it allowed us to gain skills crucial for industry and also for future collaborative assignments!