COMPENG 2SH4 Project – Peer Evaluation

Your Team Members__Lianghzhi Wang_(wangl295)___Azen Cardozo (cardozoa)_

Team Members Evaluated: Daniel Chirackal Jaavin Mohanakumar

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

Part I: OOD Quality

[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.

The header file is easy to understand. The file names and function names are straightforward, easy to follow making it is easy to assume the behaviour of the objects based on the names of the functions, but adding a couple of comments will make it easier to expect how the different functions will interact.

A note for them would be to add comments beside the function names in the header file in case there are additional functions other than what the name implies. For example, in the GameMech.h file. There are no comments there, providing a few will make it easier to understand what takes place in the objects.

[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.

The main logic is intuitive and easy to understand. The code checks for all the collisions in the player.cpp file in the moveplayer() function, making the project.cpp file shorter and cleaner. The interactions of the objects is fairly straightforward and explainable when examining the main program loop. A tip would be to add more info in the draw screen in regard to the rules that have been placed (ex. T to add more food, will not be known unless you look through the code). Another tip would be to check the input in Player.cpp file instead of the Project.cpp file.

[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:

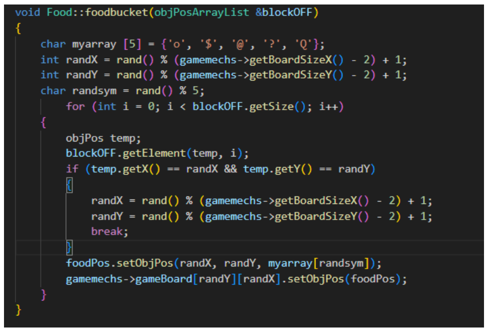
- Easier to read, better organization of code into classes and objects, and easier to maintain and update
- More secure as data can be hidden
- Allows for easier collaboration among people
- Better reusability due to the inheritance possibilities

Cons:

- Harder to debug as there are multiple files involved
- More complex to learn, especially for beginners
- C++ is slower and uses more memory due to the compilation of the multiple files

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.

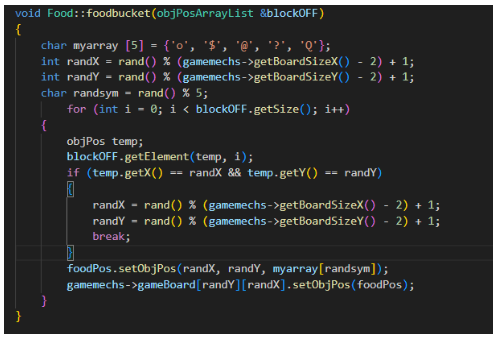


```
void Food::foodbucket(objPosArrayList &blockOFF)
{
    char myarray [5] = {'o', '$', '@', '?', 'Q'};
    int randX = rand() % (gamemechs->getBoardSizeX() - 2) + 1;
    int randY = rand() % (gamemechs->getBoardSizeY() - 2) + 1;
    char randsym = rand() % 5;
        for (int i = 0; i < blockOFF.getSize(); i++)
    {
        objPos temp;
        blockOFF.getElement(temp, i);
        if (temp.getX() == randX && temp.getY() == randY)
        {
            randX = rand() % (gamemechs->getBoardSizeX() - 2) + 1;
            randY = rand() % (gamemechs->getBoardSizeY() - 2) + 1;
            break;
        }
        foodPos.setObjPos(randX, randY, myarray[randsym]);
        gamemechs->gameBoard[randY][randX].setObjPos(foodPos);
    }
}
```

The code can use more comments. As this project is fairly straightforward and the names of the objects are self explanatory not having comments is ok for now. But having the comments will make it much easier to understand the functionality. For example, in the picture above, a short note explaining that there is a comparison between the get elements and rand elements that are created, to makes sure that food doesn't already exist there.

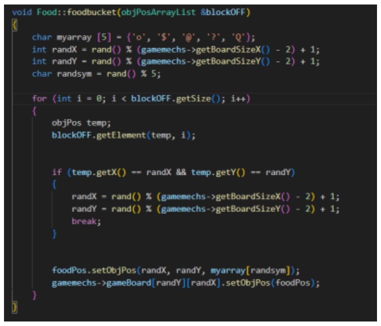
2. **[4 marks]** Does the code follow good indentation, add sensible white spaces, and deploys newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.

Before change:



```
void Food::foodbucket(objPosArrayList &blockOFF)
{
    char myarray [5] = {'o', '$', '@', '?', 'Q'};
    int randX = rand() % (gamemechs->getBoardSizeX() - 2) + 1;
    int randY = rand() % (gamemechs->getBoardSizeY() - 2) + 1;
    char randsym = rand() % 5;
        for (int i = 0; i < blockOFF.getSize(); i++)
    {
        objPos temp;
        blockOFF.getElement(temp, i);
        if (temp.getX() == randX && temp.getY() == randY)
        {
            randX = rand() % (gamemechs->getBoardSizeX() - 2) + 1;
            randY = rand() % (gamemechs->getBoardSizeY() - 2) + 1;
            break;
        }
        foodPos.setObjPos(randX, randY, myarray[randsym]);
        gamemechs->gameBoard[randY][randX].setObjPos(foodPos);
    }
}
```

After change:



```
void Food::foodbucket(objPosArrayList &blockOFF)
{
    char myarray [5] = {'o', '$', '@', '?', 'Q'};
    int randX = rand() % (gamemechs->getBoardSizeX() - 2) + 1;
    int randY = rand() % (gamemechs->getBoardSizeY() - 2) + 1;
    char randsym = rand() % 5;

    for (int i = 0; i < blockOFF.getSize(); i++)
    {
        objPos temp;
        blockOFF.getElement(temp, i);


        if (temp.getX() == randX && temp.getY() == randY)
        {
            randX = rand() % (gamemechs->getBoardSizeX() - 2) + 1;
            randY = rand() % (gamemechs->getBoardSizeY() - 2) + 1;
            break;
        }


        foodPos.setObjPos(randX, randY, myarray[randsym]);
        gamemechs->gameBoard[randY][randX].setObjPos(foodPos);
    }
}
```

The overall indentation is clean and easy to understand for the entire code. On a small note , there was a slight confusion for the indentation in food.cpp line 44. It would be better for it to align with the previous code.

An overall suggestion would be to add more new lines between each part of code as done in the pictures above. This would help with the readability quiet a bit and make it easier to look for minor and easy to fix mistakes, such indentation errors.

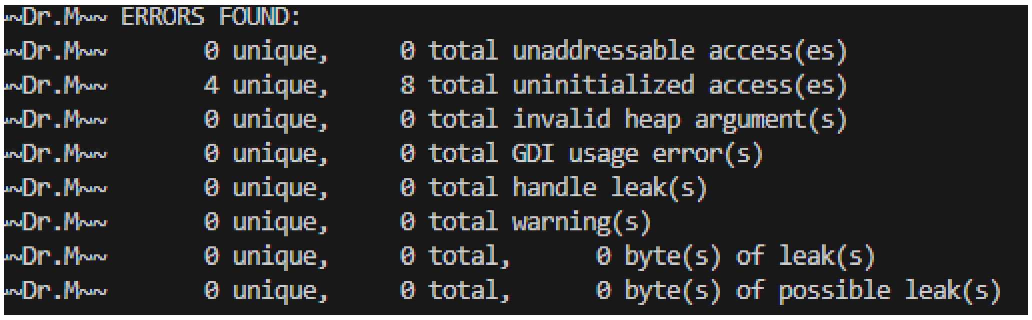
<u>Part III: Quick Functional Evaluation</u>

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 snake game is very smooth overall. The wraparound and growth of the snake works well and the game over screen functions properly as well. The random generation of food throughout the game is smooth as well, so long as only one food is generated.

One bug that was noticed was the increase in the amount of food on the screen. So, when the input to increase the amount is hit, the food that was previously generated remains in its original spot and never changes. The snake interaction still occurs, so if the snake eats the food the snake grows but the food remains in the spot after and only the newly generated food changes spots. When 3 foods are on the screen two will remain in the original spot and only the 3rd generated one changes spots. A suggestion to fix this would be to create an object list specific for the food, ex a variable in the food class, so that all the generated food can be deleted completely.

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.


```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       4 unique,     8 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total GDI usage error(s)
~~Dr.M~~       0 unique,     0 total handle leak(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
```

There were no memory leaks that were pointed out. A reason being that they had delete() statements where required, ex. GameMechanics.cpp, and used the default when the statement is not needed, ex. Food.cpp.

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.

   The experience was unique and a fun thing to learn and go about. We believe that it worked well. Working with git hub with two different individuals' stations was a bit challenging at the beginning, but after a while of using it became much easier to go about. The assigning of the iterations for each member definitely helped with progressing and getting the project done on time.