COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Mitchell Fortuna/David Cosentino

Team Members Evaluated Mitchell Fortuna/David Cosentino

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

******NOTE: the group that we were assigned to mark did not have any code in their repository. Hence, we went forth and marked our own code as was advised by the professor******

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.

Starting from the first file encountered in their program which is the bonus files of Food, it seems as though they did a decent job of importing the needed header files from different modules into their Food.h file. There does seem to be some rushed work here though with some header files having no real reason to be there such as GameMechs.h. Overall in the Food files I give it an 8/10. I gave it a decent score even though there are excess modules imported. My reasoning is that the overall concept of the file can still be discerned from the modules imported since they only made the mistake of importing this extra. (most likely a leftover from copying the player files and using it as a base for the food files…just a guess though). In their GameMechs files everything seems to be in order except for a duplicate header file occurring in both the cpp version and the .h version. Although this does not affect the program in any way besides aesthetics, it's still a bad habit as it uses unneeded computer processing power. Besides that, one mistake everything else looks good and I'm able to tell that a 2d array list composing of objpos components is to be created here just by the header files including objpos.h and objposArraylist.h. I must say they did a fantastic job keeping these program header files constrained to only those two files and no other ones like Player as it would have allowed them some ease but made the Gamemechs module much more complex. Overall, I would give them a 9/10; would be 10/10 but -1 for that header duplicate mistake. The next files are objPos. These files only have the #include guards in their header sections and do not have any other modules imported. This allows me to infer that this function is a base function for the other modules and most likely only contains a basic array, nothing complex like a 2d array (it is possible but unlikely given the name of the function). I would give this function a 10/10 in terms of the headers. This is a textbook example of how a basic building block module should be created. It has safeguards to ensure that there is no double calling it causing duplicate memory values (#inlcude guards prevent this), does not rely on other modules to work ensuring that it can stand alone, and finally it gives the consumer the creativity to use this module however they see fit without any constraints. Next up is the objPosArrayList files. The header file for this module seems to be clean with it only have the #include guards like all the modules have, the objpos.h file and a defined constant. From this alone I can interpret that this objPosArrayList is a file that creates a 2d array of the size of the defined constant which has ObjPos values. This one is straight forward, and I see no glaring errors to report so I give it a 10/10. Next up is the Player files. These files contain headers that include the objPos.h, objPosArrayList.h, and the GameMechs.h. Based off these headers I would assume that the player files create a 2d array which contains components that are of objpos type and when we account for the GameMechs.h file being included I would assume this crease the players body and runs the functions for updating their position. This one does take some inferring and previous knowledge to piece together so I would give it a 9.5/10, for a 10/10 I would have liked to see some comments at the top by the headers to allow a third party to be able to tell what the module does just by reading them. For the final file, we have Project.cpp. This file contains references to all the previously mentioned .h files. What this tells me is that this is the main file and is where the user is

supposed to draw from the different modules and use those unique functions to create a working program with OOD programming principles instead of sequential logic. Overall, I would give the entire code base a **5.75/6** since it did have some mistakes but none of them were code breaking, more just eye sores.

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.

What I like about their main logic is that they commented on almost every declaration of a variable or initialization so when reading through the code, I was able to tell what purpose this variable had and in turn this allowed me to later understand how it would be used. If we just look at the main program loop, it is easy to tell what the outcome of the function will be and to understand the loop logic. One thing that I do like is that instead of including their exit message within the while loop or with the main statements they instead placed a simple print statement in the cleanup routine since it would only be run once the while loop was exited which would only occur once the exit flag was true. This not only allowed the main logic to look cleaner but also used the while loop as a DIY if statement if you would, very clever. I like the minimalist style the coders went for in this segment and their creativity with using the loop conditions to their advantage, for that I would give a **6/6.**

```
void CleanUp(void)
{
    MacUILib_clearScreen();
    MacUILib_printf("GAME OVER\nFINAL SCORE: %d", game->getScore());   //end game message
    MacUILib_uninit();
}
```

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:

- Allows for organized ideas to take place and be implemented effectively.
- Allows for a cleaner main function.
- Makes incremental programming much easier.
- Reduces copy and paste coding.
- Allows the programmer to make specific programming choices, for example making some variables public or private depending on what they want the client to have access to.

Cons:

- Uses more memory.
- Can become overly complex and cause major errors since multiple functions in many different files could be relying on a certain function and if it fails it may take a while to determine it was the cause since it's buried in the code.
- Very easy for memory leak issues to occur here since if someone forgets to create a deconstructed, the compiler will not know where the leak came from, only that it happens.
- Harder to follow if just looking at the main program file. May have a steeper learning curve since one would constantly have to bounce back and forth between files to properly understand their functions.

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.

Comments are one of the most important tools a professional programmer can use to make software development between coworkers as seamless as possible. Upon first look, this group seems to do a job commenting each important section. Many of the critical variables and objects are described briefly yet clearly and it is easy to see what each of their purposes are. Moving on to each of the class files, the functions where their purpose may not be clear have been commented. They also took the time to comment sections in these functions where the logic is not clear, and they ensured that sections where variable names do not give a clear picture of what is going on had a short explanation. Although this group did a very good job, there were a couple shortcomings that make it hard to understand the snake expansion logic. Specifically, in the player class, there are two functions: collision and currentdir that are critical to the success of the code but are not commented at all. Both functions have only a couple lines of code to execute in each of them, but they set critical variables to specific values that dictate how the rest of the program is executed. It is possible to trace the code quickly and find what they do but someone who has no coding background will not be able to understand why these functions are here and what they do. There are other sections that do not have comments, but we feel that the variable names and the context of the program make it very clear what these switch cases do. Overall, this group does a good job of explaining section of the code that may be ambiguous to non-programmers, but they did forget to comment the purpose of two critical functions making it very challenging for anyone to discern how the game functions flawlessly. **4/5**

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.

When looking through the code, it is clear that the group took the time to make sure that formatting was adhered to and that each section could be separated from the rest. To begin, the game that is printed on the screen is clear and all the instructions are deployed with newline formatting making it easier for the user to read. Looking further into the code, this group ensured that there was plenty of indentation so that code inside for loops and conditional statements was isolated. By doing so, it was much easier to follow the progression of the code by understanding what commands were executed when the condition was met and the progression of conditions in each function. The two issues that exist here is that the exit key, which they chose to be ESC, is not explicitly stated to the user. If they want to leave the game without finishing, they have no way of leaving unless they kill their snake, which may not be possible based on its size. This issue is not that important and overall could be figured out after some time but very inconvenient. The other issue lies in how this group executed their initialization routines. When they went to set up various objects and variables, they would often just lump them all together with no whitespace. Although it is still possible to discern when each variable is created and how they are initialized using specific functions, sometimes there are 10-15 lines clumped together. Trying to understand each line can become overwhelming since many of the variables also have similar names. Overall, the code is formatted very well aside from a few minor mistakes. **3.5/4**

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)

After playing the game to maximum snake size (took a long time) and from zero snake body size it seems as though the game works as intended as once the snake reaches the maximum size allowed the program no longer

has any food being printed and the score remains constant. A non-bonus feature that would have been a nice edition would have been a win condition which would only occur if the players snake body size reached the defined constant mentioned in part 1. It seems the bonus was implemented properly, and the snake appears to be functional. Overall, I have noticed no bugs and even after doing a memory scan have found no leaks there either. A potential issue I can see in the future is their use of a global variable for their food variable. This variable appears to hold the food item data the player bumps into. I would have liked the player to use a function in the file food to return the food item that collied with the player head's data. This would allow the player to reduce the number of global variables and possibly save themselves a future headache. Overall, since everything is working properly and I have noticed no bugs and they implemented the bonus correctly, I would give this an **8/8.**

```
GameMechs* game;// Gamemechs refrence
Food* foodref; //Food refrence
Player* player; // player refrence
objPos food2; //holds values of foodbucket if a collsion occured
```

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.

No, the snake game does not cause memory leak. After much testing including various attempts at manipulating drmemory as it was running (I.e. moving the snake around to activate all the variables used and killing the snake), no leak was found. This group does everything they need to do in terms of memory management. There are deconstructors in every class where an object is initialized on the heap in a constructor and in the main program, there are multiple free statements to free up the rest of the variables and objects initialized on the heap. **6/6**

```
ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
     12 unique,   100 total uninitialized access(es)
      0 unique,     0 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)
```

Figure 1: Dr. Memory Report

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.

Mitchell Fortuna

Working with a partner allowed us to specify our efforts and made dividing the work between the two of us a simple task. There were times when it was required for both of us to work on a specific problem even though it was part of A only or B only group, but it also allowed each of us to have a better grasp of the overall code making the bonus assignment that much easier. Overall, it seemed that when we focused on only doing a certain task and only doing our portion of the code it hindered the project since it made it harder to mesh the two parts together. If I had to suggest anything, I would make it a rule that if one partner does A part in iteration 1, they must do B part in iteration 2. This would allow both parties to have a thorough understanding of the base code as well as an understanding of their partners' coding style allowing for an easier time translating their code in the future. Also, this would allow the bonus assignment to be done easier and make iteration 3 much easier to piece together.

David Cosentino:

I thoroughly enjoyed working on this project. It was definitely a challenge in the beginning to coordinate what each of us were going to do but once we figured out what the project was asking its was smooth sailing. Of course, there were bugs and errors that needed to be resolved but my partner and I understood this was a team effort and we worked together to ensure that the code was done as well as we could. I appreciated my partners commitment to the project, and he was always available to work even if he had other things to do. In summary, there isn't much that I would say was wrong with the project. Some of the classes they had us make were messy and probably could have been implemented better, but this is just preparing us for the workforce where that will definitely happen again.