COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Dennis Cao and Leo Calogero

Team Members Evaluated Sarujan and Salini

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

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.

   Looking through the header files of each object, it is easy to interpret the behaviours of the objects involved in the program and how they interact with each other. This being due to their code being put in an organized manner with private methods at the top and public methods at the bottom, sorted by data members, getter methods, setter methods, constructors and deconstructors. For example, the constructor and deconstructor for the player function are next to each other to easily distinguish its correlation and game mechs. The names of the class methods are straightforward and explain what it does with its parameters being straightforward as well as to what type of object we would want to insert. One thing which I would potentially change is to avoid using #define in the player.h file for rows and cols. This feature would likely be better as a data member in the game mechanics function. I would also comment its purpose as it isn't too clear of its usage.

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.

The main logic in the project loop is excellently done! There are very very few lines showing they implemented the object oriented design very well. They include update player direction, move player, and clear input. This allows for individuals to easy understand what is happening in the code and how objects are interacting with each other. If a change were needed to be made, it can easily be done due to the way their class methods were created. One negative however, is that I would have commented which individual features are included within this run logic. For example, within updating the player direction/moving the player, it would be ideal to know if we are also checking for collision within this function/generating new food as it may not be as obvious to someone looking at the code for the first time.

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.

C++ OOD Approach	C Procedural Design
Pros: - Much cleaner code quality - Allows for different “objects” to interact with each other more easily - Allows for overloading with constructors - Easily deallocate members on the heap with deconstructor method (won’t forget as easily) - Private members functions allow for less mistakes to be made, much easier than adding your own safe word (i.e underscores in the struct) to make sure others can’t access. Cons: - May be difficult to grasp an understanding of how different methods interact with each other/trace code from the main function back to the class code due to large amount of code required in each function. -	Pros: - Easier to understand due to the flow being all in one page/together procedurally - More control over your code due to C being at a lower level than C++ Cons: - Messier code quality - More difficult to implement “classes” due to lack of private member functions, constructors, and destructors in structs. -

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.

   The code does offer sufficient comments to help understand the functionality more efficiently. A lot of the functions in the main project.cpp file are straightforward with names such as updateplayerdir and moveplayer. These are easy to understand and do not need comments. Whenever it may not be as straightforward, there are comments to help describe what is happening/the logic behind the functions. This is evident throughout all of their code and it goes into detail on what is necessary, but never puts too much information that it may overload the reader. One thing, which was mentioned previously, is that if there is a function in the main project.cpp file that incorporates embedded features which are critical to the project, rather than having the user trace back into the code of the class, I would comment what feature is included/comment what else is done. This being specifically for the main run logic where its not explicitly stated which function includes collision.

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.

   The code does follow good indentation and follows excellent formatting for better readability. This is evident throughout their code . Through nested for loops, you can easily tell which blocks are code are encapsulated where, and it is not just a mess in which a user cannot read. The format they followed also provided good readability during project execution through proper use of new lines and indents. They used curly brackets when necessary so that it is easy to tell which block of code belongs where.

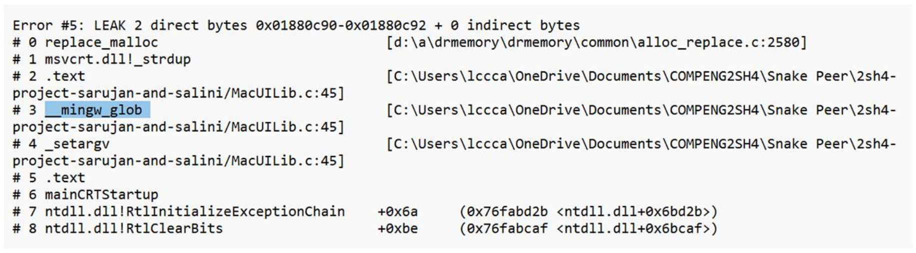
Part III: Quick Functional Evaluation

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)

The snake game offers a smooth, bug free experience. The Player positions of each objPos within the list are correctly displayed and updated as the snake grows. The score and food pos are correctly displayed. The game border is correct. Wrap-around is implemented for the player.

The snake movement is correct, can only change direction to move horizontally when moving vertically and can only change direction to move vertically when moving horizontally. The snake correctly grows with each food consumed. The snake correctly kills itself if it runs into itself. The space bar successfully exits the program. The food position is randomly generated each time. The snake body changes direction as expected. The game over message displays after suicide.

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.



```
Error #5: LEAK 2 direct bytes 0x01880c90-0x01880c92 + 0 indirect bytes
# 0 replace_malloc                          [d:\a\drmemory\drmemory\common\alloc_replace.c:2580]
# 1 msvcrt.dll!_strdup
# 2 .text                                   [C:\Users\lccca\OneDrive\Documents\COMPENG2SH4\Snake Peer\2sh4-
project-sarujan-and-salini/MacUILib.c:45]
# 3 __mingw_glob                            [C:\Users\lccca\OneDrive\Documents\COMPENG2SH4\Snake Peer\2sh4-
project-sarujan-and-salini/MacUILib.c:45]
# 4 _setargv                                [C:\Users\lccca\OneDrive\Documents\COMPENG2SH4\Snake Peer\2sh4-
project-sarujan-and-salini/MacUILib.c:45]
# 5 .text
# 6 mainCRTStartup
# 7 ntdll.dll!RtlInitializeExceptionChain    +0x6a     (0x76fabd2b <ntdll.dll+0x6bd2b>)
# 8 ntdll.dll!RtlClearBits                   +0xbe     (0x76fabcaf <ntdll.dll+0x6bcaf>)
```

The program causes no memory leak. The 2 bytes of leakage is caused by the compiler and not by their program as highlighted. The array list created on the heap within the objArrayList class is deallocated within the class deconstructor. There are no heap data members created within the objPos and Game Mechanics Classes. Within the player class, the objPos array list is correctly deallocated within the classes deconstructor. The instance of the GameMechanics class created on the heap within the main program is deallocated in the cleanup routine. The instance of the player class that is created on the heap within the main program (Project.cpp) is also deallocated in the cleanup routine. Due to the above, there is no memory leakage being caused by the program.

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.

   Collaborating in software development was an excellent experience. Not only was the work divided evenly, in the event of a bug or error, we had two sets of hands working to fix the issue rather than one. Furthermore, being able to work in parallel at the same time really allowed for greater efficiency. One thing that may not have been working though is the mistiming of gitpushing, as sometimes there would be an error when one person modifies a file at the same time as the other.