McMaster University

Comp Eng 2SH4 Project Peer Evaluation

Your Team Members Fatima Sarfraz Sophia Vetzal

Team Members Evaluated Sankavi Caylia

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

Part I: OOD Quality

1. **[5 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 group has followed good naming conventions and C/C++ naming standards. Thus, it is easy to tell what each object may do. The header files are neat and not cluttered with unnecessary objects. Since the group mainly stuck to the naming conventions listed in the recommended workflow, which our group followed as well, we are familiar with what is happening in most of them. The only confusing thing is that the group defines a board width and height in the Game Mechanics, but never seems to use it in the main logic, instead defining global variables in the Project.cpp with different values. Another possible source of confusion is that in GameMechs() default constructor, speeds are mentioned but then are never used in the main program.

2. **[5 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 group displays a good use of OOD by calling on the various classes where appropriate. As a result, the main logic of the program is not cluttered. They have showcased good knowledge of pointer reference by assigning appropriate pointers to each class such as myGM to Game Mechanics and playerPtr to the Player class. By doing so, they are easily able to call on the functions in each class, allowing them to maintain good readability and organization.

   The group does however miss out on a few opportunities to call on a class. For example, in the Project.cpp they have globally defined new heights and widths for their game board and reference these variables (namely "BOARD_WIDTH" and "BOARD_HEIGHT") in Run Logic and Draw Screen. Instead they could have used the getBoardSizeX and getBoardSizeY member functions from Game Mechanics. Another issue with the overall cohesiveness, is that in Game Mechs, the width and height are defined to be different (30 and 15) than in Project.cpp (where they are defined to be 36 and 18).

   One other negative feature in the main logic in the main program loop is that some functions are mixed up within the methods. For example, MacUILib_printf() is used in RunLogic() and Cleanup(). It is better practice to set a variable to true and then use an if statement in DrawScreen() that way the program is more organized.

3. **[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.

   Pros:

- Better organization and less clutter in main logic
- Since everything is split up into different source codes, it is easier to understand what is happening and what each class and their member functions do
- Inheritance/Polymorphism: able to reuse, modify and extend existing classes. You are easily able to perform new functions without having to build a lot on the raw code since the pre-made classes already provide most of the foundation

   Cons:

- After doing so many labs and PPAs in C, we became accustomed to all the C conventions and regulations and since the classes and objects were such a new topic, the learning curve was a bit steep
- Chances of memory leakage are greater as you allocate more and more on the heap and may sometimes forget to either check the memory report/delete the variables
- It is hard to wrap your head around all the classes and member functions sometimes. Especially with so many member functions, you tend to forget that a function exists and miss the opportunity to use it.

Part II: Code Quality

1. **[4 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 offers quite a few comments and thus it is easy to understand what each function is doing. One good thing I noticed is some codes have been commented out and these were there debugging processes (they've written in the comment above that it was for debugging purposes). It's nice to see a bit into their thought process and see how they tried overcoming possible bugs. One short coming is that a few parts of their code could have been easily condensed. For example, in their update player position logic, their switch case is a bit lengthy as they have tried to include all the restrictions (i.e. when moving up can't move down). This makes it harder to understand that you cannot move in the 180 degree opposite direction. To improve this they could have put a simple if statement such as in the case of 's' being pressed, if the direction is not set to UP only then will myDir be set to DOWN. Furthermore, I would also advise to the group in general to really understand the functions of each class and keep them in mind while patching everything together in the main logic so you don't miss an opportunity to deploy one of the member functions of a class and your code will become less bulky and more concise.

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 is easily readable with proper indentation, appropriate white spaces and newline formatting. There are always indents after an opening curly brace and spaces between different constructors or functions. Overall, the code is neat and not difficult to follow. One shortcoming is that they did not delete their debugging messages. This is especially apparent as they print "debugging message" in drawScreen.. This could have been deleted to give the code a more finished feel. When submitting a final code, especially in a professional setting, it is good practice to get rid of unnecessary comments, such as past debugging to make the code as tidy as possible.

Part III: Quick Functional Evaluation

1. **[6 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 self-collision does not always work, and it does not always display the lose Flag message. This may be due to the exitFlag and the loseFlag being set to true when there is a self-collision. It appears that checkSelfCollsion() excludes the tail of the snake given that the for loop on line 131 in Player.cpp uses < instead of <=. < means that the loop will reach one less than playerPosList->getSize(). I would recommend changing the for loop conditions as well as commenting out setExitFlag and then monitoring any changes in the behaviour of the code.

Other than those changes, the code runs smoothly and works as expected!

2. **[4 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.

   There appears to be no memory leak in the game. A destructor is used anytime new is employed, which removes the potential for memory leak. playerPosList, myGM, and playerPtr are all deleted, and all use the new keyword.

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.

   Collaborated software development had a few pros and cons. On the positive side, it was beneficial to bounce ideas off one another and reduce the workload of the project. The main con of collaborative coding was trying to share copies of our codes. We stopped being able to git push our codes and had to manually upload our updated codes which was time-consuming and was prone to human error. If there were more debugging tips on why we were unable to git pull or push, it would've been very helpful as I know many people struggle with this. Overall, I felt very supported through this project, and it was a great first experience.