COMPENG 2SH4 Project – Peer Evaluation

Your Team Members Hamza Abbas Umar Javaid

Team Members Evaluated Rehan Mohan

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 at the header files, for the most part, it is easy to understand what each object is meant to do, except in the player class. Beforehand, in the gameMechs, objPos, and objPosArrayList, it is very easy to tell the purpose of each class and what it is meant to do. However, some aspects don't come together in the player class. For example, they are missing checkSelfCollision, checkFoodConsumption, and increasePlayerLength methods. Without these, snake game is much more difficult to implement (in this case they aren't implemented at all). Apart from that, you can see that all other files are included in the player class, which makes it simple to identify that all others are meant to "come together" in the player class to implement the snake game. They were on the right path and have correct OOD practice (constructor and getter methods that make it clear what their purpose is), but without certain features (i.e checkselfcollision) the game will be incomplete. Maybe something they could improve on is commenting for some methods why something is being passed in, to give others an even better understanding of what is going on.

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.

   Reading through their main program, they do a great job keeping it clear and organized. They make great use of OOD design, which is seen in their very concise Runlogic function (only 3 lines of code). However, since they are lacking appropriate methods in the player class (i.e increasePlayerLength), their code in the initialize contains a portion of the logic that would normally be in their increasePlayerLength method. Apart from missing aspects of the game, they clearly use objects in the game, and I can understand how everything interacts due to the efficient naming of the methods. For example, in the runLogic it is clear to see they are using updatePlayerDir and movePlayer methods to move the player object around the screen. In the main function, these positions generated from runlogic are used in the drawscreen function to

properly print them onto the board (use methods such as getPlayerPos, and getFoodPos to print things onto the board). Overall, their main program loop is very easy to follow through.

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

   - C++ OOD utilizes classes to perform the tasks (dirty work done in the background). This makes the main function extremely easy to follow and understand whereas the procedural design made the main executable file confusing, not allowing others to easily read and understand what is going on.
   - Adding new features or editing existing ones is a lot more straightforward and manageable.
   - Due to private features, it helps to ensure unintended modifications to existing code doesn't happen.

   Cons

   - Requires many different files (cpp and headers) to work together. Which can quickly result in errors if done incorrectly (more difficult learning curve for developer).
   - Due to many different files being used, errors/bugs can be a lot harder to track down and solve for those inexperienced programmers (our first time using OOD, so it was harder to understand)
   - More memory intensive due to having to manage objects.

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.

   This execution of the project does deploy very sufficient self-documenting coding style. Without even looking at the logic, just by looking at the name of the methods in each class you can easily identify what the method is going to do, even without commenting. All the method names are very self explanatory so looking at them called in the project.cpp makes it easily understandable what is going on. While the classes are either commented sufficiently or properly self-documented, the main project.cpp has very little comments which makes it hard to understand some intentions. For example in the drawscreen function, they call for objPosArrayLis* playerbody = myplayer->getPlayerPos(). While people familiar with pointers can somewhat interpret the purpose, it would be more clear if they could outline what this initialization is for (what its pointing to, why a pointer is used, etc). So, to improve I would say to make more clear comments on the Project.cpp so anybody reading can easily follow along and understand (even with limited coding knowledge), specifically in the DrawScreen and Initialize functions.

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.

   Yes, the following code follows proper indentation and adds sensible spacing, making it easy to read and understand what is going on. In each header file, getter elements and constructors are grouped, making the code easy to understand. Things are indented and spaced nicely, following proper format and allowing things to be easily understood. In the draw screen function, things seem a little bit condensed making it somewhat more difficult to read. To improve readability I would add a little bit more spacing and make sure else and if statements are consistent (some had the { bracket on the same line as the if statement, others had it on the next line). Apart from that very nicely organized.

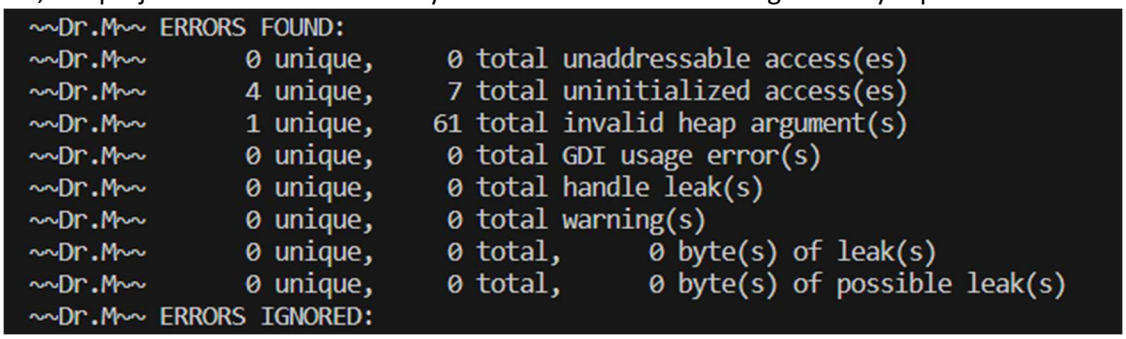
Part III: Quick Functional Evaluation

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)

   checkselfcollision, checkfoodconsumption, and increaseplayerlength methods, No, the snake game provided does not give a smooth playing experience. Although they are able to generate food, a gameboard, and a player onto the screen, there are many bugs in the game. For example, when first loaded up, it starts with the single player at the start of the screen, correctly randomly generated food, and proper game board, however when you start moving that's when the errors arise. To start, as soon as a movement key is pressed, it inserts a tail 4 times without hitting a single fruit and stays that way even after colliding with a fruit. Also, colliding with the fruit also doesn't add anything to the length of the snake, it just generates a new position. Movements are also extremely buggy (pressing opposite movement does an action when nothing is supposed to happen). Because each bug is "isolated" due to classes, they can each be analyzed individually without messing with each other. To debug, I would look at their player class because that's where most errors stem from. Clearly there are issues in the updatePlayerDir and moveplayer logic (looking at their code, they are missing instead they tried to include all that in the moveplayer which checks out as to why so many errors occur). To debug I would recommend making sure correct parameters are passed in properly, and setting up gdb debugger to check outputs for the movement and collision logic.

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 project.exe leaks no memory as indicated in the following memory report:



Rehan and Mohan properly deallocate memory from the heap and no memory leakage occurs in the code. In their code, they correctly use destructors in the objPosArrayList, player, and gameMechs classes, freeing memory from the heap and resulting in no leakage. They also properly freed memory in the CleanUp section, clearing both myGM and myPlayer objects used in the executable using delete (deallocating memory).

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.

   During the project, we felt as though the provided workflow for both developers made the overall work experience very smooth. We found following the workflow, we could focus on our own deliverables and in the end, we can combine it together. Me and my partner also commented our code throughout, which helped the other person easily understand what the other was doing. The most difficult part was sharing the code with each other without accidentally erasing the others work. Overall, the process was extremely smooth.