McMaster University

COMPENG 2SH4 Project – Peer Evaluation

COMPENG 2SH4: Principles of Programming

Team Members: Joseph Natale, Nate Hunter

Team Members Evaluated: Sydney Sochaj

McMaster University

Part I: OOD Quality

---

1. Managing and understanding the differing objects through their designated names was quite easy given Sydney's code, with accurate and/or understandable header files given to them. This was done through full or abbreviated descriptions of the intent of the object being in its header file, as seen through this example of the “foodpos” object. This header refers to the object tracking the coordinates of the generated food item, which can be easily gathered from the name, with “pos” being short for position. There was a lack of redundancy or similarities in headers, allowing for not only easy interpretation but also a lack of confusion and mistaking one object with another. Overall extremely comprehensible, with a good amount of effort displayed to denote each objects behaviours, as well as allowing for deduction through understanding its purpose, of how it will interact with other objects, as seen through this name as an example: “hasFoodGeneratedThisFrame()”.

2. Through examining the logic in the main program loop, the logic is comprehensible, however, some key portions of the logic are commented out, as well as there being a significant amount of logic in the “DrawScreen” function, with it also containing functions to generate food. It is somewhat more difficult to follow the process of how their objects interact together, through the commenting out of key functions such as one that verifies if the player has reached the object, as well as their primary logic being displayed in the DrawScreen function. The Runlogic function serves only to update and move the player, however that is easily comprehensible, albeit a bit separated. Meanwhile in the drawscreen function, it is possible to follow the logic and the objects interacting with each other, as it uses and interacts with the “myPlayer” and “myGM” objects, in order to retrieve data about the player's position, food position, and other imperative information. Through these lines of code we can see a demonstration of how the objects interact and work together. However, that brings us to the negative features of the code. Due to the commenting out of key functions, there is a somewhat lack of coherence in the run logic function, as well as the logic then having to be called in the drawscreen function in order to attempt to generate the food. There are also some cases of repetition in the DrawScreen function, such as when the variable “drawn” is declared both inside and outside the loop.

3. Pros:
    - One pro to the c++ approach is its promotion of modularity, through its use of encapsulation to increment code.
    - Through the proper use of headers and file management, the code is much more easy to understand and organise, and as such much more friendly to outside viewers.
    - It is much easier to debug using the c++ approach, as issues are much more separated from each other and can be more easily singled out.

- Has the ability to inherit code, from which reduces redundancy.

Cons:

- Multiple files will have to be managed in this OOD approach compared to the singular file used in C procedural design approach of previous projects.
- The code can unintentionally be led to be more complex, from which issues can arise and need to be managed.
- Using this approach compared to C, one needs to manage their accesses much more, and make sure they are not used at the wrong times, or else it may lead to errors, encapsulation issues, and more.
- It can be more difficult to comprehend the flow or process of the code, as it is calling functions from other files, and as such in order to understand the flow you must go back and forth from differing files.

McMaster University

Part II Code Quality

---

1. The code establishes numerous comments and formatting to aid user interpretation, with many if not most loops and if statements, as well as a wide variety of variables containing comments beside them, denoting their purpose or descriptions. The steps to each function are incremented well, and the thought process behind each function can be much more easily understood and followed through the comments. All of these combined with the previously mentioned in part 1 headers and names, allows for the code to be easily understood and distinguishable, with you understanding where each line of code is meant to be and what functions it's a part of. One recommendation I would add would be, that despite the multitude for singular purposes being explained by comments, there could arguably be more describing how different lines work in tangent with each other, or the overarching intentions of a function could be described with comments more.

2. The code is well formatted, with proper indentations, brackets and white spaces, to more clearly portray the code and demonstrate how it is intended to be interpreted. Through this, when reading a line of code, it is easily distinguishable if it is inside any statements, loops, functions etc.. Overall the formatting of the code is very well organised and neat, and there are not many criticisms or recommendations to be had with its layout.

McMaster University

Part III: Quick Functional Evaluation

---

1. When the program is run on windows it will not run properly. After cloning the repository we changed all the required lines to define windows in the macuilib and make files. Even after changing all of these lines and running the program, the game displays the lose screen overtop of the game and printed out a random large score. It also would increase the player size on every move and would never remove the tail. After reviewing the code we realised that the score variable and the lose flag variable weren't defined in gamemechs but were being modified in the class. This lead to a random number being defined for score, and loseflag automatically set to true. We defined score to 0 and defined lose flag to false at the beginning of the code. This allowed the game to be played properly, but when you lost the game screen would stay printed overtop of the losing screen. The issue causing this was because of the the loop for continuing the run logic and draw screen was only dependant on the exit flag. We changed the loop conditions from "while(myGM->getExitFlagStatus() == false)" to "while(myGM->getExitFlagStatus() == false && !myGM->getLoseFlagStatus())". That way when the game ended in a loss, it would stop the drawscreen and run logic and move into the cleanup function. Also, the part of the code that displayed the losing screen was included in the drawscreen function instead of the cleanup function. We moved the contents over to the cleanup function so that it would be displayed after the while loop was complete. After making all of these changes the code worked fine on windows with no errors. When the code is run on linux, the code works perfectly fine directly after cloning the repository. The reason for this is because windows and linux define variables differently without declaration. On linux the score variable is set to 0 by default instead of a random memory address.

2. After doing a memory report with valgrind, the report showed no memory leak caused by the game. The report showed 0 bytes of definite loss and 0 bytes of indirect loss. All new calls onto the heap are properly freed leading to no memory leakage. Memory is initially allocated in the code onto the myGm and myPlayer pointers through the gamemech and player classes. However it is properly cleaned up and released in the CleanUp function, where the delete operator was used on both in order to release them and prevent memory issues.

McMaster University

Part IV: Your Own Collaboration Experience

Working on code collaboratively was a new experience with pros and cons. One of the biggest positive aspects was having two people to review the code when debugging. Sometimes it is very hard to find a small error in your code, especially after coding for a long time. Having a second person review your work opens up opportunities for someone else to catch something that you may be missing during debugging. Another positive is being able to split up different parts of the code and completing it at the same time. This drastically changes the amount of time needed to complete the full program. One of the hard parts of coding collaboratively is not having matching codes until the code is pushed to github. Sometimes we wouldn't realise the other group member had already updated the code and we would end up working on a section of the code that didn't need to be changed. This means that every single change needs to be communicated between members which can be difficult when you're constantly editing code. Working collaboratively is definitely a good asset, but needs practice to perfect.