COMPENG 2SH4 Project – Peer Evaluation

Your Team Members	Yousif Fadhel,	Yusuf Rashwan
Team Members Evaluated	Akashdeep Singh,	Adam Jasinski

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.

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.

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.

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.

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.

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)

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.

Part IV: Your Own Collaboration Experience (Ungraded)

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.

ANSWERS

Part I: OOD Quality

[Question 1]

Following inspection of this groups' header files it is clear that the contents (naming, organization, documentation) allow for easy interpretation of what the structures are intended to do.

Objpos.h : Example of header file in the repository.

```
1  #ifndef OBJPOS_H
2  #define OBJPOS_H
3
4  class objPos
5  {
6      public:
7          int x;
8          int y;
9          char symbol;
10
11         objPos();
12         objPos(objPos &o); // copy constructor
13         objPos(int xPos, int yPos, char sym);
14
15         void setObjPos(objPos o);
16         void setObjPos(int xPos, int yPos, char sym);
17         void getObjPos(objPos &returnPos);
18         char getSymbol();
19
20         bool isPosEqual(const objPos* refPos);
21
22         char getSymbolIfPosEqual(const objPos* refPos);
23 };
24
25 #endif
```

Using the above example some positive features observed in this header file include the comment on line 12 that highlights the difference in constructors in a way that is more apparent than by just analyzing the passed parameters. Furthermore, the naming conventions all are easily followable and allow anyone reading the code to interpret its behaviour.

An example of this from this group's Objpos.h file is the "isPosEqual" function that passes the objection position (objPos) and the reference position (refPos) while returning a Boolean. Based off of line 20 alone, it is clear that this function will check to see if the reference position is equal to the objects position and will return a Boolean (True or False) depending on the outcome.

Objpos.h : Example of header file in the repository.

```
48
49  bool objPos::isPosEqual(const objPos* refPos)
50  {
51      return (refPos->x == x && refPos->y == y);
52  }
```

In terms of critiques, a few very minor adjustments could be made in the self-documentation aspect of the “Player.h” header file. Some comments remained in the code despite having been resolved later on through out the group’s development process as seen in line 22 in the below screenshot.

Player.h: Screenshot of line 22, outdated documentation.

```
15
16      public:
17          enum direction {UP, DOWN, LEFT, RIGHT, NONE};  // This is the direction state
18
19          Player(GameMechs* thisGMRef);
20          ~Player();
21
22          objPosArrayList* getPlayerPos(); // Upgrade this in iteration 3.
23          void updatePlayerDir();
24          void movePlayer();
25
26          bool checkSelfCollision();
27
```

Despite this miniscule critique, the analysis of the header files reveals a well-structured and concise modularization, demonstrating a thoughtful application of object-oriented design principles. The clear delineation of class responsibilities, proper encapsulation, and absence of unnecessary dependencies collectively contribute to the program's robustness and maintainability.

[Question 2]

The main program loop is decently designed. Since the member functions are well-named, there was no problem understanding what was happening step-by-step and how they interact with one another. One suggestion we have is moving the getters for the flag statuses in RunLogic, as well as their setter. To us, that makes more sense since GetInput should simply get the input. Another thing we’d like to point out is the repeated check of the flags after each function within the while-loop of main. While it is not inherently wrong to do so, one check should theoretically be enough. We removed all except one check, and realized it caused a delay in exiting the game, so there is a reason they are repeated, but there should be a way where one is sufficient.

[Question 3]

Pros :

- Reusability : C++, unlike C, supports the creation of reusable classes and objects drastically reducing redundancy.
- Encapsulation: Provides encapsulation for data hiding and enhanced security. Helpful to ensure nothing is accessed on accident and helps protect code when distributed.
- Modularity: As touched upon in question 1 C++ encourages modular design through the use header files, making it easier to manage and maintain code.

Cons:

- Complexity: Due to the OOD approach it can sometimes be difficult to stay organized with the various different header files, and keeping track of public, private, and protected structures. PPA3, was more straightforward in terms of what code was for us to edit and we were able to instantly work on functions directly rather than worry about class development and consistency.

Part II: Code Quality

[Question 1]

The code offers sufficient comments. Each function has a comment broadly describing what it does, and the more complex functions go more into depth for each section within said function. The coding style is also mostly self-documenting; however, we would suggest that some variables should have their names changed. For example, the Player constructor implies a temporary position, when in reality this is really the starting position of the snake's head.

```
4   Player::Player(GameMechs* thisGMRef)
5   {
6       mainGameMechsRef = thisGMRef;
7       myDir = NONE;
8
9       //creating the default snake
10      objPos tempPos;
11      tempPos.setObjPos((mainGameMechsRef->getBoardSizeX()-2)/2,(mainGameMechsRef->getBoardSizeY()-2)/2,'*');
12      playerPos = new objPosArrayList();
13      playerPos->insertHead(tempPos);
14  }
```

Although, even without these changes, the code was quite understandable.

[Question 2]

The code mostly follows good indentation, while adding whitespaces and newlines where required. We say mostly because while the majority of indentations consist of 4 space characters, there are some with 3 (which really is no problem; 3 or 4 are fine), but the GameMechs.cpp file has simply 2 space characters. This can somewhat hinder the ability to read the code, but since the GameMechs.cpp file is not code-heavy, it was not really a problem. Consistency in the indentations would be nice, but we know it can be difficult with a group whose members have different coding styles. The whitespaces and new lines were adequate, and there was no section of the code that had an absurd length, nor were there blocks of code that were difficult to look at.

Part III: Quick Functional Evaluation

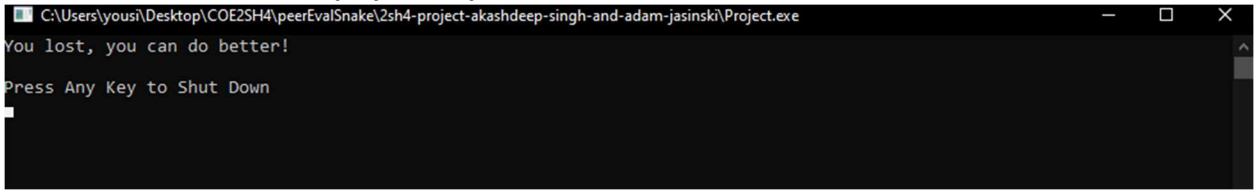
[Question 1]

After playing the given group's snake game various different times no bugs were encountered. The gameplay is responsive, and all features function as expected. This suggests a well-implemented and thoroughly tested codebase, contributing to a positive user experience. Evidence of the user experience is documented below:

Start Screen: Player begins in the middle of board, score starts at 0, food is displayed.

Mid-Game, scoreboard updating appropriately

Game Over Screen: Will display when your snake collides with itself.



[Question 2]

After running the memory profiler “drmemory ./project.exe” in the terminal and producing the report it is evident that there is no memory leak in this code that can be attributed to the actual work of the programmers. This can be concluded through observation of the “ERRORS FOUND:” section of the report where the only instance of memory leak applicable is due to the established format of the assigned project and not the group’s implementation.

drmemory Report:

```
===========================================================================
FINAL SUMMARY:

DUPLICATE ERROR COUNTS:
        Error #   1:     16
        Error #   2:     16
        Error #   3:      7
        Error #   4:      7
        Error #   5:      7
        Error #   6:      7
        Error #   7:      6
        Error #   8:      6

SUPPRESSIONS USED:

ERRORS FOUND:
      0 unique,     0 total unaddressable access(es)
      9 unique,    73 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)
ERRORS IGNORED:
     20 potential error(s) (suspected false positives)
        (details: C:\Users\yousi\Desktop\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.28312.000\potential_errors.txt)
      1 potential leak(s) (suspected false positives)
        (details: C:\Users\yousi\Desktop\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-
Project.exe.28312.000\potential_errors.txt)
     39 unique,    39 total,  10374 byte(s) of still-reachable allocation(s)
        (re-run with "-show_reachable" for details)
Details: C:\Users\yousi\Desktop\DrMemory-Windows-2.6.0\drmemory\logs\DrMemory-Project.exe.28312.000\results.txt
```

PART IV:

[Your Own Collaborative Experience]

This collaborative software development format allowed our group the opportunity to understand the intricacies behind delivering organized code that is sufficiently self documented to ensure through the progression of every iteration, the program remains easily digestible to

both partners. This format also allowed for the delegation of parts between partners. The project manual suggested for each partner to work on an iteration, thus we adopted this approach allowing for an increase in our efficiency when completing this project. Working collaboratively on an assignment such as this one also gave us insight into what a career could look like in this field at a company where multiple people will be expected to be able to understand and contribute to an established project. In summary, the collaborative software development format not only deepened our understanding of organized and self-documented code but also provided valuable insights into teamwork dynamics, project efficiency, and the real-world expectations of contributing to established projects, offering a glimpse into the collaborative nature of careers in the field.