COMPENG 2SH4 Project – Peer Evaluation

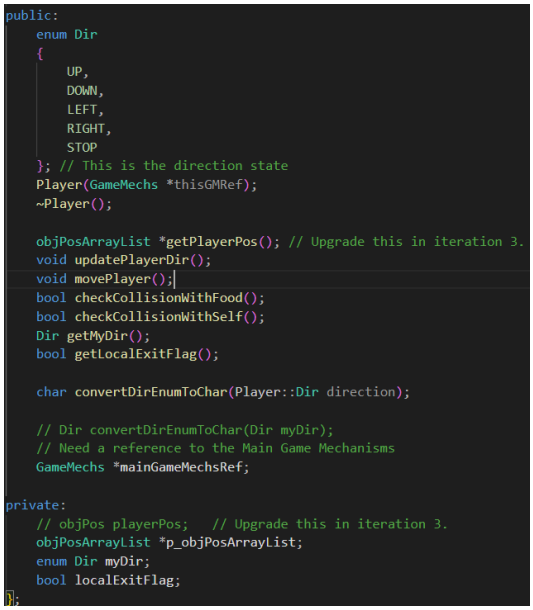
Your Team Members	Stanislav Serbezov	John Sarga
Team Members Evaluated	Zerui Chen	Shijun Gao

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, it is easy to interpret the possible behaviours of the objects involved in the program. Everything is named well, and it is easy to read most times. There are some places where more spacing and organization should have been used to help readability. An example is the Player Header file:



```
public:
    enum Dir
    {
        UP,
        DOWN,
        LEFT,
        RIGHT,
        STOP
    }; // This is the direction state
    Player(GameMechs *thisGMRef);
    ~Player();

    objPosArrayList *getPlayerPos(); // Upgrade this in iteration 3.
    void updatePlayerDir();
    void movePlayer();
    bool checkCollisionWithFood();
    bool checkCollisionWithSelf();
    Dir getMyDir();
    bool getLocalExitFlag();

    char convertDirEnumToChar(Player::Dir direction);

    // Dir convertDirEnumToChar(Dir myDir);
    // Need a reference to the Main Game Mechanisms
    GameMechs *mainGameMechsRef;

private:
    // objPos playerPos;   // Upgrade this in iteration 3.
    objPosArrayList *p_objPosArrayList;
    enum Dir myDir;
    bool localExitFlag;
};
```

There is no grouping of similar functions and no spacing to show the different parts of the program each function is a part of. Here is a better version I organized:

```
public:
    Player(GameMechs *thisGMRef);

    ~Player();

    enum Dir
    {
        UP,
        DOWN,
        LEFT,
        RIGHT,
        STOP
    }; // This is the direction state

    Dir getMyDir();
    char convertDirEnumToChar(Player::Dir direction);
    void updatePlayerDir();


    objPosArrayList *getPlayerPos(); // Upgrade this in iteration 3.
    void movePlayer();
    bool checkCollisionWithFood();
    bool checkCollisionWithSelf();

    bool getLocalExitFlag();

    // Dir convertDirEnumToChar(Dir myDir);
    // Need a reference to the Main Game Mechanisms
    GameMechs *mainGameMechsRef;

private:
    // objPos playerPos;   // Upgrade this in iteration 3.
    objPosArrayList *p_objPosArrayList;
    enum Dir myDir;
    bool localExitFlag;
```

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.

Some parts are easily readable, such as the if statement that checks if the game is still running and the logic that checks if the snake collides with the food or itself. Some other parts are a bit harder to understand, such as the initiation of the playerPos variables. There is a playerPos variable that is declared but never initialized, and then used in the code, which isn't very readable. Also, they named the list containing the info on the snake body "pobjPosArrayList", which isn't very descriptive and causes some confusion. A name like "playerBodyPos" or something similar would have been much clearer and more descriptive.

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.

   <u>C++ OOD Approach</u>

   Pros:

   - The modularization of code makes it easier to organize the code into classes so its easier to manage
   - It allows for code reusability
   - Able to protect data by restricting access to class members

   Cons:

- Harder to get familiar with
- Can become pretty complicated if not designed properly.
- Adds unneeded complexity to simple programs

C Procedural Design

Pros:

- Procedural coding was much more straightforward and it might be easier to understand
- Easier to work with since its more intuitive

Cons:

- Reusing code is more challenging and makes code very redundant
- When moving to a larger project, procedural programming will get very confusing and difficult to maintain

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.

   Most of the code is commented sufficiently, with simple comments that give a general idea of what happens in the code. There are a couple of places in the code where a more descriptive comment would be helpful. For example, in the function removeHead() shown in the screenshot below, there is a comment mentioning that the head element will be removed. However, someone could look at the for loop and be confused on what is happening. One suggestion is to add a comment before right before the for loop, for example on line 53 that mentions that you are shifting the elements in the array to the left. This could remove some confusion for the people reading the code. There are also no comments specifically describing what happens in the switch cases. We think it would be a good idea to always add a more specific comment before any loops or switch cases so that the reader can understand the functionality easier.

```
49  void objPosArrayList::removeHead()
50  {
51      // Remove the header element
52      if (sizeList > 0)
53      {
54          for (int i = 0; i < sizeList - 1; ++i)
55          {
56              aList[i] = aList[i + 1];
57          }
58          sizeList--;
59      }
60  }
```

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 is easy to read. What I mean by this is that the code is not heavy on the eyes. The indentation is correct everywhere and newlines are used properly for better readability. I like how the parts of the code that are related are written as one block and then white spaces are used to separate the less related items. There are some places where more spacing and organization could have been used to help with readability. There aren't many blocky parts in the code. The only blocky part of the code is in the DrawScreen shown in the screenshot below. Since it is all printing statements it isn't really an issue but it could have been shortened by using the cout statements more efficiently. For example, lines 156-159 could all be written in one line, shortening the number of lines in the code. Lines 156-159 could be written as:

```
cout << "Board Size:" << boardX << "X" << boardY << endl;
```

```
151     else
152     {
153         cout << "You score:";
154         cout << pPlayer->mainGameMechsRef->getScore() << endl;
155         cout << "========= DEBUG MESSAGE ========" << endl;
156         cout << "Board Size:";
157         cout << boardX;
158         cout << "X";
159         cout << boardY << endl;
160         cout << "Direction:";
161         cout << pPlayer->convertDirEnumToChar(pPlayer->getMyDir()) << endl;
162         objPosArrayList *pobjPosArrayList = pPlayer->getPlayerPos();
163         objPos playerHeadPos;
164         pobjPosArrayList->getHeadElement(playerHeadPos);
165 
166         cout << "Snake Head: " << playerHeadPos.y << ", " << playerHeadPos.x << std::endl;
167 
168     }
```

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).

One bug is that the food gets printed directly to the right of its original position immediately after it is eaten. This issue causes the game board to get 1 character longer in the x direction wherever this character is printed. This issue is caused by both the draw and generate items function. The draw function should only print a food character when there is no snake body at that position to fix the food being printed on the snake's body. The way to fix the food being moved one to the left and the gameboard getting wider is to check if the snake head and the food collied right after the snake is moved and before it is drawn. Then you call the call the generate food function immediately instead of 1 frame later in the current implementation. Generally, you should look at the logic that causes generate food to be called and make sure it is correct to fix the issue.

Another bug is that initial score is a garbage value. This is caused by the initial score not being initialized. They should look at the GameMechs constructor and make sure they are initializing score correctly to fix this issue.

They didn't output a message when the user quits the game. This doesn't directly affect the gameplay experience, but it can cause confusion if the player did not mean to quit the game and it closes unexpectedly for no apparent reason. Additionally, they didn't quit the game when the user lost, which causes some confusion as well.

A small nitpick is they left their debugging messages in the final version. This slightly disrupts the gameplay experience because it causes unnecessary confusion among players by displaying information they don't need nor have any control over.

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.

   The Snake Game does not cause a memory leak. There are only uninitialized reads. At the end of the DrMemory Report 0 bytes of leaks were reported.

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.

   Overall, this collaborated project experience was good. We made sure that we were communicating every step of the way. First, we defined the roles and responsibilities that each of us would have and then we split off to work on the code individually. We made sure to share our progress updates and any challenges we faced with each other which made the development process much smoother. One issue we ran into was when we were pushing or pulling to git. Git was nice to use because we were able to work independently and then merge work however, it was annoying whenever we ran into an issue that we didn't expect when pulling or pushing. When that happened, we had to figure out how to fix it. It would have been nice to have a video that could explain possible issues on git and how to fix them. Other than that, the project went really smoothly and we are happy with our outcome.