COMPENG 2SH4 Project – Peer Evaluation

Your Team Members : Max Fang, Jacky Nam

Team Members Evaluated: Nikolai, Ross

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.

ObjPos.h

- Behaviour of the objects/methods are easily interpreted
    - For example, methods like SetObjPos() are easy to interpret, representative of their function
    - Member variable name are also easy to interpret, representative of what they are
        - Only suggestion would be using a more descriptive name of the variable passed by in setObjPos(objPos o); and objPos(objPos &o);

- Variables like x, y, and symbol could be declared private to control how they are accessed and modified
    - These private variables could then have public getter methods to access
        - this would further extend encapsulation in your project

GameMechs.h

- Header guards are good to prevent namespace pollution
- The include lines that include ObjPos.h and ObjPosArrayList.h allowed me to easily interpret that there is a relationship between these classes
    - Ex: boardSizeX, setExitTrue(), getScore all are easy to interpret

- Variables are declared private, and have their respective getter method
    - Variables and methods are also very easy to interpret and are well named

- Really really cool start screen and end screen, awesome idea
    - Encapsulated into a function for readability, great

- Not an issue at all for this project, but declaring a namespace (ex. std) is generally looked down upon by developers

Player.h

- In the DIR enumeration, "STOP" is not used in your switch case
  - Game is fully functional, just something to point out

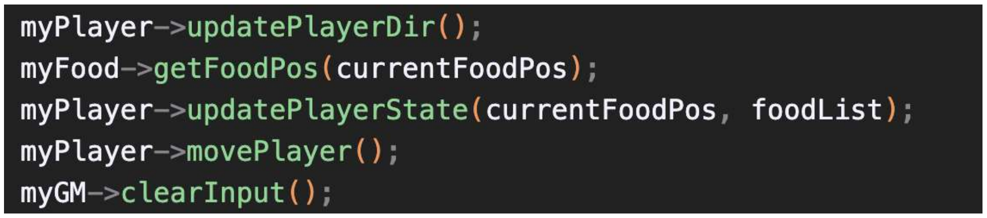
- Switch case included Capital letter counterparts
  - Great idea if player has caps lock on

- Just like the other header files, members and methods are very readable
  - Food check, consumption, and self collision all encapsulated in their own function within game mechs, great
  - Good use of the private scope to prevent members from

Food.h

- Good use of the private and public scope to keep members safe
  - Getter methods implemented and are well named, easy interpretation
  - Food being in another class/header makes your project very modular, great

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.
   - **Simplistic and coherent with naming and order. It is easy to follow which header file is being referenced and function definition provides enough information to suggest the processes going on.**



```
myPlayer->updatePlayerDir();
myFood->getFoodPos(currentFoodPos);
myPlayer->updatePlayerState(currentFoodPos, foodList);
myPlayer->movePlayer();
myGM->clearInput();
```

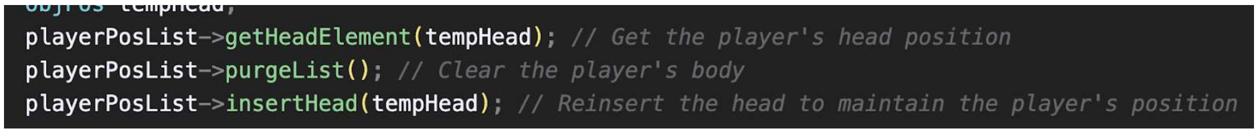
   - **Minimal/no comments, nitpicking here, could implement comments that state processes occurring depending on the state of the game or about user found errors in a location can be traced back to a certain header for inspection.**

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.
   - **Lots of modularized objects that bypass the need for initializing variables and reducing excess clutter which is 'blackboxed' away.**

- **With the right naming conventions, formatting and labelling, it becomes very easy to understand what each code block does and it is fairly easy to trace where information is being processed/accessed/changed.**
- **When done incorrectly/inefficiently/incoherently, the OOD approach becomes a mess to fix and read for others. Things are likely to break and tracing the break usually involves changes in each header that rely on each other.**

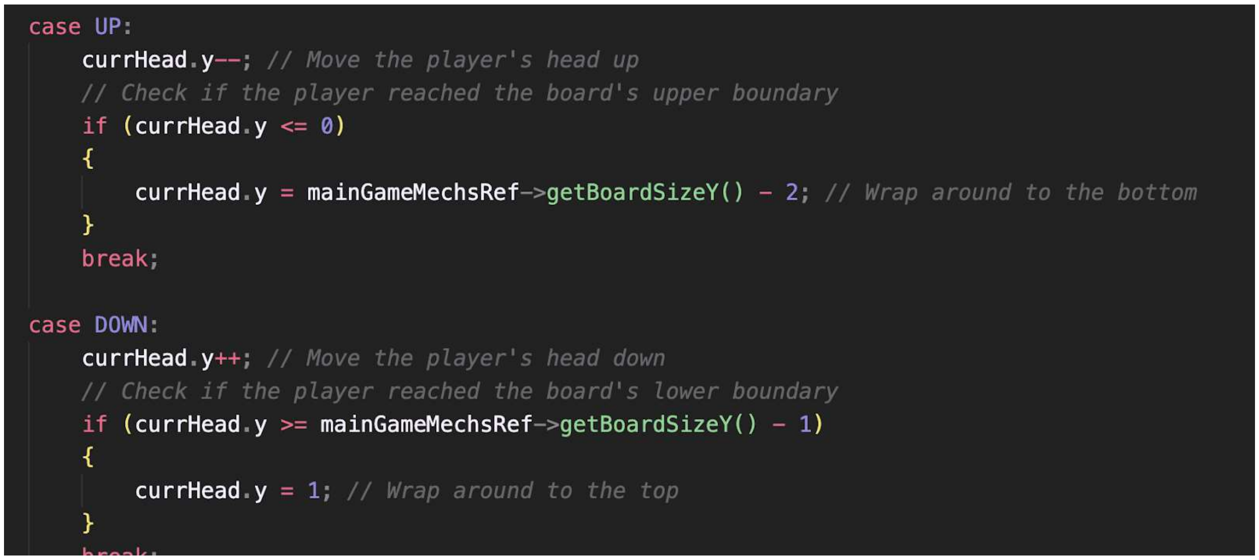
Part II: Code Quality

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.
    - **Very well documented, maybe excessive in parts where the function name is self-explanatory.**
        - **present in header cpp files such as**



```
playerPosList->getHeadElement(tempHead); // Get the player's head position
playerPosList->purgeList(); // Clear the player's body
playerPosList->insertHead(tempHead); // Reinsert the head to maintain the player's position
```

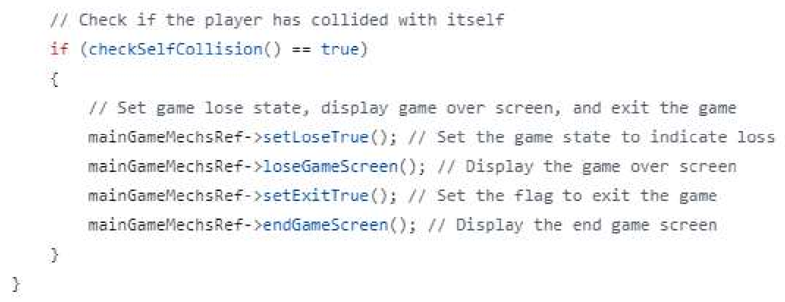
    - **Comments could be on above the functions or separated from implemented code in areas where it appears cluttered. One comment for ‘x’ processes may be sufficient if they do the same process with different inputs.**
        - **present in player movement**



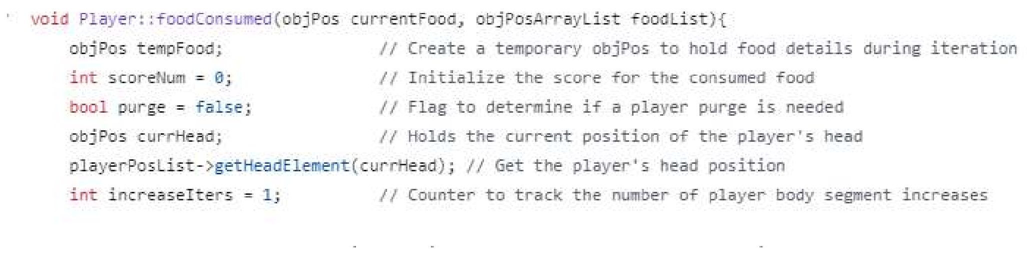
```
case UP:
    currHead.y--; // Move the player's head up
    // Check if the player reached the board's upper boundary
    if (currHead.y <= 0)
    {
        currHead.y = mainGameMechsRef->getBoardSizeY() - 2; // Wrap around to the bottom
    }
    break;

case DOWN:
    currHead.y++; // Move the player's head down
    // Check if the player reached the board's lower boundary
    if (currHead.y >= mainGameMechsRef->getBoardSizeY() - 1)
    {
        currHead.y = 1; // Wrap around to the top
    }
```

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.

- **Yes, aside from the mentioned comment clutter in specific areas the code is formatted clearly both in the main Project cpp and header files.**
- **Little improvement could be made in specific header files where the readability is affected.**
    - **Example below shows a comment in each line of Player.cpp where the function calls are self explanatory, comments could be adjusted to be placed at the top and one concise sentence.**



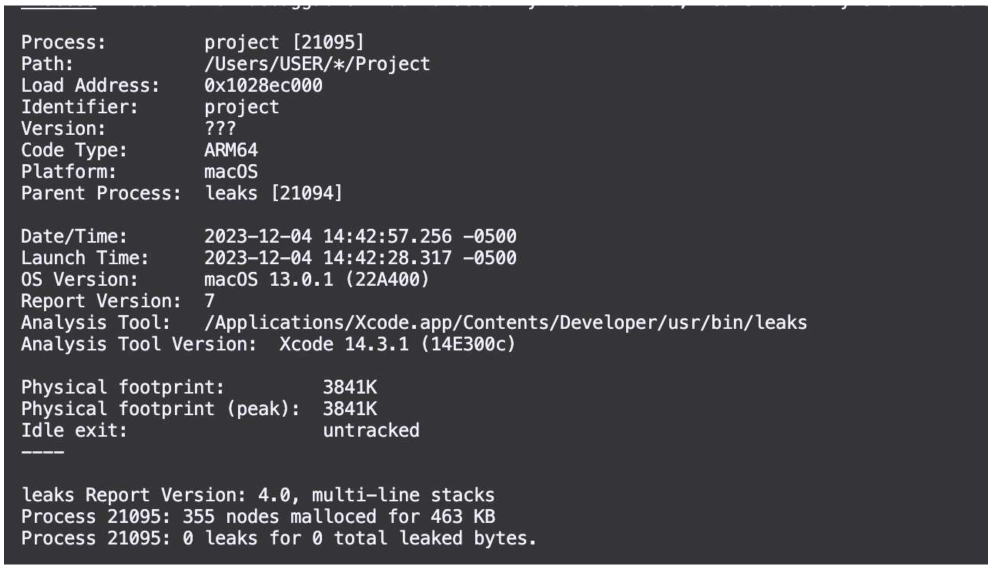
```
    // Check if the player has collided with itself
    if (checkSelfCollision() == true)
    {
        // Set game lose state, display game over screen, and exit the game
        mainGameMechsRef->setLoseTrue(); // Set the game state to indicate loss
        mainGameMechsRef->loseGameScreen(); // Display the game over screen
        mainGameMechsRef->setExitTrue(); // Set the flag to exit the game
        mainGameMechsRef->endGameScreen(); // Display the end game screen
    }
}
```



```
void Player::foodConsumed(objPos currentFood, objPosArrayList foodList){
    objPos tempFood;                // Create a temporary objPos to hold food details during iteration
    int scoreNum = 0;               // Initialize the score for the consumed food
    bool purge = false;             // Flag to determine if a player purge is needed
    objPos currHead;                // Holds the current position of the player's head
    playerPosList->getHeadElement(currHead); // Get the player's head position
    int increaseIters = 1;          // Counter to track the number of player body segment increases
```

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)
    - **Player movement delays/input not recognized sometimes when moving diagonally quickly in succession.**
        - **Could be a delay set between inputs/draw screen/logic processing.**
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.
    - **No leaks observed after playing the game collecting every type of food presented and testing all the features presented.**
        - **includes sudden game stops, finishing, starting by collecting the special food**


```
Process:         project [21095]
Path:            /Users/USER/*/Project
Load Address:    0x1028ec000
Identifier:      project
Version:         ???
Code Type:       ARM64
Platform:        macOS
Parent Process:  leaks [21094]

Date/Time:       2023-12-04 14:42:57.256 -0500
Launch Time:     2023-12-04 14:42:28.317 -0500
OS Version:      macOS 13.0.1 (22A400)
Report Version:  7
Analysis Tool:   /Applications/Xcode.app/Contents/Developer/usr/bin/leaks
Analysis Tool Version:  Xcode 14.3.1 (14E300c)

Physical footprint:         3841K
Physical footprint (peak):  3841K
Idle exit:                  untracked
----

leaks Report Version: 4.0, multi-line stacks
Process 21095: 355 nodes malloced for 463 KB
Process 21095: 0 leaks for 0 total leaked bytes.
```

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.

1. Collaboration was fun with the right partner that shares a similar mindset. Solving bugs that came up when merging code was frustrating yet enticing to see another perspective in achieving the same result.
2. Navigating shared repositories and not knowing what changes were made (not applicable to us), could be unmotivating and could cause distress if program breaks after implementing partner changes.
3. Collaboration provided a better workload, much easier to manage sections of code and knowing other parts will be finished while working on individual portions. Conversely, this may add stress, uncertain if the other parts will be implemented correctly.