Assignment 4: Overview

I hope you will learn something about how to collaborate with the people and utilize the code already existing in the world. Also, be familiarized yourself with how to do Object-Oriented Programming!

Modify the skeleton code and complete the implementation of `MyPlayer`

- Do not modify any files other than `MyPlayer.h` and `MyPlayer.cpp`
    - If you know what you're doing, you can change other files for testing purposes. However, note that you will only submit the two files.
    - You may also want to change main.cpp for your testing purpose, but you will not submit
- You CAN NOW use any C/C++ library functions and classes, including STL.
- Do not create any other header/source files. All of your implementations should be included in the two files, "MyPlayer.cpp" and "MyPlayer.h"
- All the implementations should be executed without any runtime/compile error, e.g., your implementation should check all corner cases that may cause segmentation faults.
- If you find a bug in the provided source code, please send an email to me.

Important!

- Do not try any cheating like code copying and modifying. We can really identify them.
- However, this time you are allowed to share the source code only for the purpose to battle your strategy against another student's strategy in advance.

Note: This description describes only a part of the provided program. Please understand the details of the lecture that I gave on Nov 11.

Due

Submit your implementation before Dec 6, Sunday, 11:59:59pm, to LMS.
We DO NOT allow late submission. It will be very strict.

Important!

- If you want to see the preliminary results, submit your implementation before Nov 29, Sunday, to LMS. I will run the small league for the source code submitted by then and give the results back.
- You have multiple You will have a chance to submit your implementation before Dec 6, Sunday, 11:59:59pm, to LMS.
- We DO NOT allow late submission. It will be very strict this time.

Submission

Submit the source(.cpp) and header (.h) files that have your implementation.

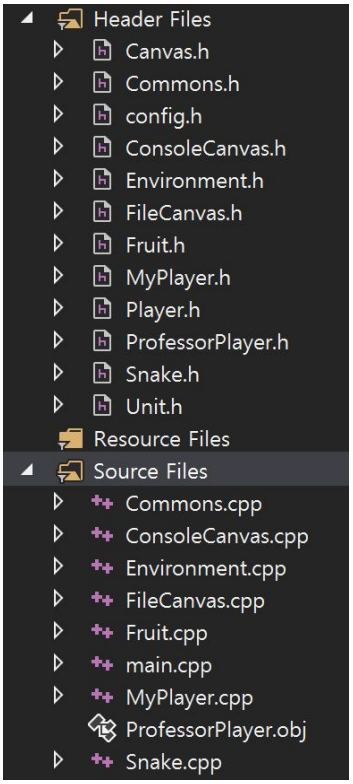
- Compress two files, “MyPlayer.cpp” and “MyPlayer.h”, in a single zip file, naming “hw3_YOURSTUDENTID.zip”
- DO NOT submit any other files. They will be ignored.

Assignment Specification

What should I do?

Compile:

1. Make the project with the provided code package to compile
    a. In the visual studio, create a console program and add the source code and an object file (which implements the professor’s two strategies)
    b. It should look like this:



2. Once it successes, you can watch the game of a single round --
MyPlayer (very naive strategy) vs. Prof. NeverDie Strategy (advanced strategy)

Rule of the game:

- In the 15x15 map, it generates two snakes randomly at two positions, (3,3) or (11,11).
- It also generates two fruits at random positions.
- Player 1 takes the first turn, and then Player 2 takes the next turn.
- For every move, the player can decide the next direction of the snake. But, there is a restriction in the selection - the snake cannot rotate in 180 degrees. For example, it is heading to the north, it cannot rotate toward the south.
- If the snake hits the wall or the body of the two snakes, it will die.
- If a snake takes a fruit, it grows. The taken fruit is removed from the map, and a new fruit is generated in a random position again.
- If one of the snakes crashes or the maximum number of turns (300) reaches, the game ends.
    - The alive snake wins if one snake dies.
    - If two snakes are both alive till the maximum number of turns, the winner is determined based on the length of the snake.

Implement your strategy that selects the direction of the snake:

1. Do something to initialize in the class constructor of MyPlayer
    a. Currently, it sets the size of the map
2. Your main task is to implement "`MyPlayer::selectDirection`" member function
    a. The current implementation has a naive rotating strategy. It selects the north, west, south, and east direction repeatedly for every turn.

```
Direction MyPlayer::selectDirection(
        int turn, Snake* player_snake, Snake* enemy_snake,
        std::vector<Fruit*> fruits) {
        switch (turn % 4) {
        case 0:
                return DrtN();
        case 1:
                return DrtW();
        case 2:
                return DrtS();
        }
        return DrtE(); // when case 3
}
```

3. Function parameters
    a. int turn
        i. The turn number.
    b. Snake* player_snake & Snake* enemy_snake
        i. You can get the information for each snake using these classes
        ii. For example, the following function returns an STL vector that contains all body positions the snake occupies from the head to the tail.

```
virtual std::vector<Pos> getPositions() const override;
```

            1. Each position is defined with its x and y coordinates.

iii. For example, the following function returns the head direction

```
Direction getDirection() const;
```

1. The direction is defined with its moving position of x and y coordinates

iv. For example, by adding the head position with its direction, i.e., player_snake->getPositions()[0] + player_snake->getDirection(), you can calculate the next position of the snake head when keeping the same direction in your turn.

v. For a more complete description of all public member functions, refer to the source code comments provided.

c. std::vector<Fruit*> fruits

i. The pointer variable of the two fruits

ii. For each fruit, you can check the position using the `getPositions` function. It will always return a vector that has only one element.

Grading

- The game league will happen using the configuration specified in config.h, i.e., in the 15x15 map, two fruits randomly created, three-sized snakes at default, and 300 turns at maximum.
- The full league will be held! For example, if there are 70 students, I will run the league games of all pairs of 70x70.
    - For each student pair, I will run 10 rounds and decide the winner.
    - The order of players will be randomized for every round, i.e., who will take the first turn.
    - The fruit positions will be also randomized for every round.
- This time, I will try to give grading with a wider range than the previous assignments.
    - If you win the naive policy that MyPlayer initially implements, you will get the 10 points (out of 25) at least.
    - If you win the Prof. FruitTaker, you will get the points, 15 out of 25, at least.
    - The rest of the score will be determined based on the ranking in the league.

Some extra comments

- This assignment is probably difficult in that you should understand how the provided source code works. So, please expect a *good* amount of time to digest the source code.
- You should also think about how to come up with a winnable strategy. An example is to take a small number of fruits and wait until the enemy dies. But, the counter-strategy of this one is to take at least one more fruit and wait until the enemy dies.