Welcome to our Tankwar!!

Here, we want to first show how to compile and run this game,and then introduce deeper the pattern and paradigm we use, after that, we'd like to share some isuues and bugs' solving strategy we use during coding.

How to compile and run this tankwar game?

- compile the game:

  The whole game consists of three files named layer1.h, layer2.h, and layer3.cpp. So you need to first get all three files in the same project in your compiler and add this following sentence in CMakeLists:

  ```
  add_executable(layer3 layer3.cpp layer1.h layer2.h)
  ```

  Then you are totally ok to run the layer3.cpp to enjoy the game.

- run the game:

  The whole game uses commandline argument so that you need to set some information before running it.

  ```
  ./layer3 --mode=PVP --initial-life=5
  ```

  Here is a sample set before running the file, the mode setting and initial life settig are compulsory, or the game can't run successfully. You can print:

  ```
  ./layer3 -h
  ./layer3 --help
  ```

  in the program to see some more detailed settings.

  ## I/O format and log file

We choose to make use of the char array to visualize the map every round,so after every turn, the player will immediately see the layout of the game with several instructions for player to input some certain integars to control his/her own tank.

As for the log file, the whole game will automatically be saved in a log file under the default folder when you run this program. However, there's some problems with the log-file function， so for your tailored log file, you can change the code in layer2.h 457:

```
log.open("tankwar.log", ios::app)
```

The default file name is "tankwar.log", but you can replace it with any other name you want so that the game can be output to your tailored file.

Layered Architecture Pattern we use

Just as the name of each three files, we decompose the whole program into three layers, and each serves different functions and usages.

- layer1:

  1.basic definitions of the `Class Tank`, `Class Bullet`

  2.declarations of their functions.

- layer2:

  main functions setting of the program:

  - **for tank**:

    1.function to initialize

    2.function to turn and move tanks

    3.function to judge collision

    4.function to shoot bullet

  - **for bullet**:

    1.function to move

    2.function to judge collision

  - **for the whole game**:

    1.function to judge winning

    2.function to shrink the border

    3.function to generate AI behaviour

    4.function to plot the map

- layer3:

  the main program to run the game:

  1. the procedure to read in command-line argument and adjust them to the initial situation
  2. the different setting of different mode(PVP,PVE,DEMO)

Object-Oriented Paradigm we use

In this game, two classes are enough to realiza the OOP, but in order to ease the setting and change of position and direction, we difine three classes each named `Vec`, `Tank` and `Bullet`.

- **`Vec`**

  ```
  class Vec:
  int x;int y;      //two int numbers representing x and y axis
  ```

- **`Tank`**

  ```
  class Tank:
  Vec Pos;            //instant location
  Vec direction;      //instant direction
  int life_point;     //instant lifepoint
  void initialize();  //function to initialize tank
  void move();       //function to move
  void turn();       //function to turn
  ```

- **`Bullet`**

```
class Bullet:
Vec Pos;           //instant location
Vec direction;     //instant direction
```

Issue and Bug we met during coding

1.Since the procedure of judging whether the position of tanks and bullets are the same would be repeated several times throughout the process, it’s too tedious and awkward to repeat judging whether the values of `x` are the same and whether the values of `y` are the same.

Therefore, we redefine the “==” and “>” operators so that whether the tanks crash and whether the bullets hit the tanks could be judged simply with “==” just as ints:

```
bool operator == (Vec a)const {
      return (x == a.x) && (y == a.y);
```

2.In the log file function,it appears that double-using the same iterators when plotting the map will cause the whole program to accidentally end unexpectedly,it may be due to the excessive restortation of the memory.And when we use two different iterators to plot the map either on commnad window and argument, the results appear as usual.

However, the filename in the command-line argument can't be taken as a string and serve as an input of an function to print into a tailored log file. It may be due to our method to plot the map, so our game will automatically save every game you play into a document named "tankwar.log", and user can only change the file name by directly changing the code as mentioned in I/O part.

The simple AI algorithm we use

Since the rules in this tank war game is quite simple,so the AI can have a relatively good operation according to three judgements,with decreasing priorities:

1.Tank crash judgement

Since the quickest way to end this game is collision between tanks, so if the player's tank is close enough and its life isn't higher than AI's, then chasing and crashing on purpose is a good way.

While if AI's life is lower, the smartest way is running away.

2.Bullet collision judgement

AI should detect the three location it may step on next round by calculating which location will have bullet, and delete this way in next round's movement.

3.Central judgement

Since the border will shrink, so it can't be smarter to stay in the middle of the map. If the previous two judgements all have no feedbacks, than AI will choose to automatically move towards the center.

Here is our Tankwar game's brief introduction.Thanks for your patience!

And wish you have fun in this game!!