VG101 Lab 7 - Tank War

Author: Lu Haorong

OJ Platform: http://vg101.sometimesnaive.xyz/ (Can only be accessed within SJTU, you may use SJTU VPN if you are outside SJTU)

If you have any question, please contact: ancientmodern@sjtu.edu.cn, or via WeChat/Feishu

Special thanks to **Wang Kaibin**, the TA of VG101 in FA2019 and FA2020, and **Cai yunlong** from JI IT department

Master of MATLAB, C ---> RoboMaster

GOAT (The Greatest of All Time). This is the only word you can come up with to describe your robot. You firmly believe that under the same control, your robot, the crystallization of your wisdom and hard work, will easily destroy all other competitors in this RoboMaster Contest. But there is still one most crucial gem before the final victory, the driver. As a picky perfectionist and rationalist, you trust that human intelligence cannot be worthy of your robot. There seems to be only one choice: an awesome, powerful AI (Artificial Intelligence).

Obviously, it's too dangerous to directly test the AI on your priceless robot. ***You refer to the classic Tank War game and design an AI battle game to simulate the RoboMaster Contest.*** Now it's just a matter of recruiting game participants and cannot make them aware of the connection between this game and the RoboMaster Contest. After some thinking, you come up with a brilliant idea……

You suggest this game to one VG101 TA who is worrying about designing the last lab. The incompetent and naive TA treats you as his hero. Without any hesitation, he accepts your proposal and promises to give a bonus to those who perform well in the game. This is exactly what you want: ***Creating the GOAT robot AI*** through the involution among students!

As a careful guy, I believe you must have found that the bold italicized part above is what you need to do in reality :)

Background && Rules

Tank War is a classic computer game. In this game, each player controls a tank and they compete to become the last survivor. This time, you are going to implement a simple version of Tank War and write an AI to compete with other students. In this LAB, you will learn how to build and use data structures in C++, and have a taste of game programming and traditional AI programming.

To make your life easier, we create following rules for the Tank War,

- Two players battle in a spare map.
- Each player controls a tank.
- The tank moves 1 **meter/turn**.
- Each turn, the tank can choose to **move left, right or forward**.

- The tank shoots a bullet **every three turns** (starting from the first turn), and the bullet will be generated 1 meter ahead of the tank's position. (The bullet has the same direction as the tank, and it is generated after the tank moves in its turn, otherwise the tank will be shot by itself)
- Bullet **moves straightly** at the speed of 2 **meter/turn**.

- Tank have 5 **life points**.
- Once a tank is hit by a bullet, 2 **life points** will be deducted.
- To make your life easier, just assume that the bullet moves after the tank moves since this will avoid geometry calculations. Note that each turn there are **three possible cases** where the bullet may hit a tank in your calculation.
- Once a bullet hits a tank, it will not go further but explode.

- Initially, the the size of the map is 20×20. (A tank can go outside the map)
- The map shrinks by 1 block to the center every 16 turns (the first shrink occurs in the $16th$ turn), namely the map size will be $20 \times 20 \rightarrow 18 \times 18 \rightarrow 16 \times 16$ ...
- As long as a tank is outside of the map, **one life point** will be deducted each turn.

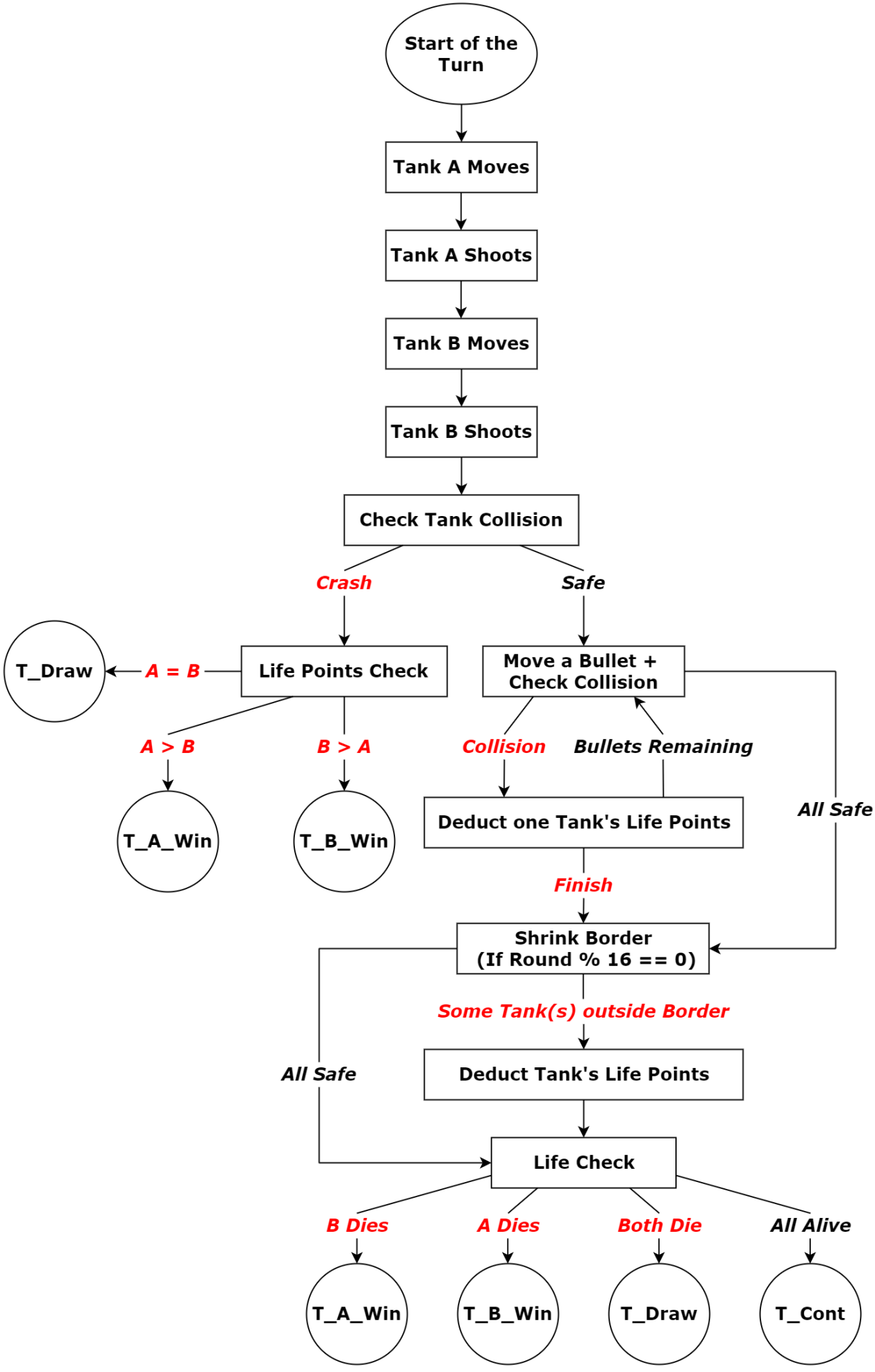
- The tank whose life points are first deducted to 0 will lose.
- If the life points of two tanks are deducted to 0 in the same turn, they have a draw competition.
- When two tanks crash, the tank with high life point wins.
- To make your life easier, each turn the calculation takes place after two tanks move.

Task 1

Given `Lab7_Starter_Files.zip` on canvas, implement a Tank War game based on the provided structure.

To implement the Tank War, you need to **design a data structure to store the positions and directions of tanks and bullets**. Then you can think about how to update the states of tank and bullets in each turn.

Below is a sample flow chart to show what happens in each turn,


Start of the Turn
Tank A Moves
Tank A Shoots
Tank B Moves
Tank B Shoots
Check Tank Collision
Crash
Safe
T_Draw
A = B
Life Points Check
Move a Bullet + Check Collision
A > B
B > A
Collision
Bullets Remaining
All Safe
T_A_Win
T_B_Win
Deduct one Tank's Life Points
Finish
Shrink Border (If Round % 16 == 0)
Some Tank(s) outside Border
All Safe
Deduct Tank's Life Points
Life Check
B Dies
A Dies
Both Die
All Alive
T_A_Win
T_B_Win
T_Draw
T_Cont

One thing that needs to be clarified is that after a bullet is spawned, **it will move two meters in this turn.** So in the flow chart above, the **"Move a Bullet" part includes the bullet that is just generated this turn.**

You can define your own data structure and write the code in `lab7.cpp`, and it's better not to modify other related files because you can only submit `lab7.cpp` to the OJ platform later. The other files will be provided by us.

Example I/O module is provided in `main.cpp`.

Sample Inputs from `stdin`:

```
0 0 19 19 2 0
2 0
1 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
2 0
0 0
0 0
0 0
1 2
1 2
0 0
1 1
1 1
0 0
0 0
0 0
0 0
0 0
0 2
0 0
0 0
2 0
0 0
0 0
0 0
0 0
0 0
2 0
0 0
0 0
```

```
0 0
0 0
0 0
0 0
```

Corresponding outputs in `stdout`:

```
Tank A Wins
```

In the first line of input, `0 0` and `19 19` are the initial positions of tank A and tank B respectively, `2 (Right)` is the initial direction of tank A, and `0 (Left)` is the initial direction of tank B.

The coordinates here are slightly different from the Cartesian coordinates we commonly use. **It is more like the index of a two-dimensional array in C/C++.** So the top left point is $(0, 0)$, and the bottom right point is $(19, 19)$.

The following input lines indicate the actions of two tanks in each turn. **Here 0 means moving forward, 1 means turning left and moving forward, and 2 means turning right and moving forward.**

For the detailed definition of direction and action, you can check the `enum` definitions in `lab7.h`

```
enum Direction {
    D_Left, D_Up, D_Right, D_Down
};

enum Move {
    M_Forward, M_Left, M_Right
};
```

Note

- The `draw()` function is optional this time. It's just used for the `stderr` part on the **"OJ platform - Task 1 Submissions"**. If you want to use this function to debug your program, or experience how to use `stringstream` in C++, you can implement it, otherwise you can just output anything you want.
- OK, I find that `draw()` function is helpful when you debug your program. To make your life easier, I provide my implementation of `draw()` function. But note that the implementation actually depends on your data structure to store tanks and bullets. **So you need to make some small modifications to adapt it to**

your own data structure.

- ```
  string draw() const override {
      ostringstream result;
      for (int y = -5; y < MAP_SIZE + 5; y++) {
          for (int x = -5; x < MAP_SIZE + 5; x++) {
              char cur = ' ';
              Vector2<int> curPos(x, y);
              if (x < border || y < border || x >= MAP_SIZE - border
  || y >= MAP_SIZE - border) cur = '-';
              if (TankA.pos == curPos) cur = 'A';
              if (TankB.pos == curPos) cur = 'B';
              for (auto &bullet: bullets) {
                  if (bullet.pos == curPos) cur = '*';
              }
              result << cur << "|";
          }
          result << "\n";
      }
      result << "\nA: " << TankA.life << ", B: " << TankB.life <<
  endl;
      return result.str();
  }
  ```

- It's hard to illustrate all you need to do in the manual. You can read the comments and codes in `lab7.h` and `lab7.cpp` for detailed information and instructions. All you need to do are specified by ***TODO***.
  - Among all functions in Task 1, `initialize()`, `move()` and `turn()` are the most important functions to run the Tank War naturally;
  - `draw()` and `getMap()` are functions that help you debug or help us judge the correctness of your program;
  - `addBullet()` and `setBorder()` are functions that let you control the Tank War in God Mode.

Task 2

Given `Lab7_Stater_Files.zip` on canvas, write an AI for your Tank War in `MyBrain` class. An AI framework is provided in `MyBrain` class, and a reference of `MyGame` object is already obtained, so your AI can have access to your current game through the interface you provide in your `MyGame` class.

Submit your AI to http://vg101.sometimesnaive.xyz/, and fight for glory!
  ```

Note

- If you are eager to battle with others, you can only implement the `initialize()`, `move()` and `turn()` functions in `MyGame` class, other functions only serve for Task 1.
- **If you find your account is occupied by other, please contact me ASAP!**
- Note: It's OK if your classmates also want to participate in our Tank War, but please remember to inform us! We will add his/her student ID to our backend database.

Guidelines on OJ

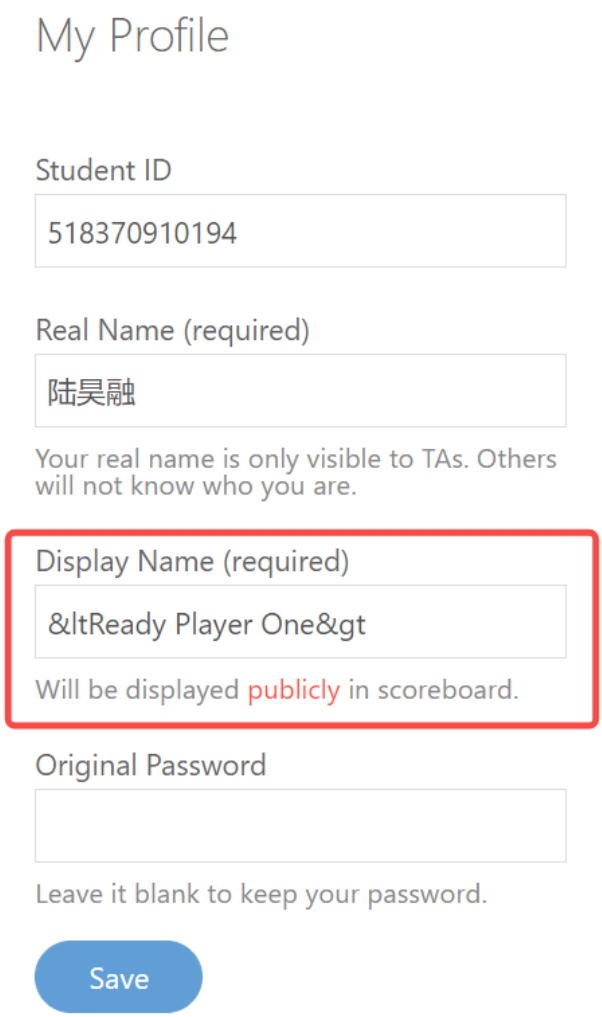
- The OJ does not have a clear separation between "sign up" and "sign in". The first time you click **Login** , you need to enter your student ID (do not enter other's student ID!) and set the password, this is like **create an account**, and now you should see your name on the scoreboard. Then you need to click **Login** again, enter your student ID and the password you just set, and this time it's really **login**.
- To submit your AI, first **Login** and then click **Submission - Submit New Brain**, and **paste all the content in your** `lab7.cpp`. Choose the C++ versions you want to use, and then click "Submit". If everything goes fine, you will see the **Effective** flag, which means currently your tank is controlled by this brain.

Security and Privacy Concerns

- Your passwords is stored in encrypted form on the backend, so don't worry if TAs or others can access your password.

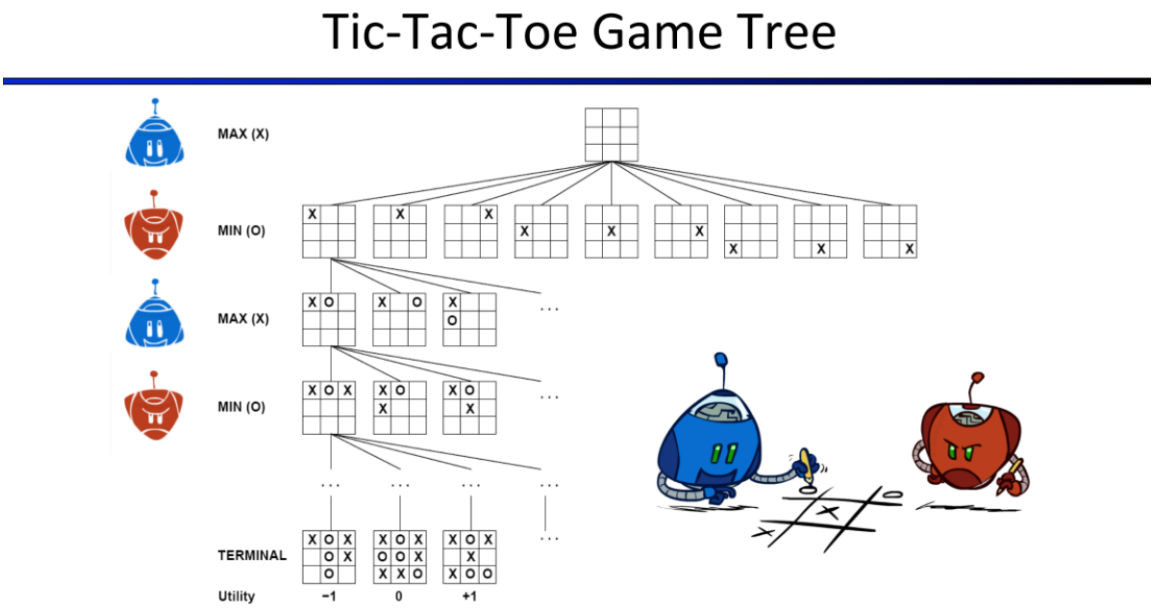
```
> db.user.find().pretty()
{
        "_id" : [redacted]
        "studentId" : "518370910194",
        "password" : "/G1Zq9RGwiE0c3+rmaVtxXeS1eGFUCP8E7YErA49l+8=",
        "realName" : "陆昊融",
        "dispName" : "&ltReady Player One&gt",
        "admin" : true,
        "win" : 69893,
        "lose" : 0,
        "draw" : 0,
        "score" : 3559.9017804349037,
        "compiler" : "c++17",
        "student" : true
}
```

- Also, you can change the display name if you do not want to show others your real name.



- If you know something about HTML, you can find that in the above screenshot, the < and > is automatically replaced by `&lt` and `&gt`. This mechanism is used to prevent potential XSS attack.

Hints

- You do not need to understand the class inheritance and virtual functions in `lab7.h`, it simply enables you to write your own structures without changing `lab7.h`.
- A simple 2D vector class `Vector2` is provided. You may use it to help implement your Tank War and AI.
  - The `template<typename B = int>` enables it to accept every type, even the `class` or `struct` defined by yourself. For example, you can write `Vector2<string>` to let it use `string` type. If you do not specify the type (`Vector2<>`), it will use `int` as the default type.
  - We also overload some basic operators to make it more useful. Don't be scared by the "strange" definitions. Here is a simple example to help you better understand them,
  - ```
    Vector2<int> a{1, 2}, b{2, 3}, c; // According to the constructor, c.x == 0, c.y == 0
    c = a + b;    // c.x == 3, c.y == 5;
    c = a * b;    // c.x == 2, c.y == 6;
    c -= (b - a); // c.x == 1, c.y == 5;
    ```
- How to write an traditional AI?
- **Game Trees**
  - Each node illustrates one state of the game, and has its utility
  - Higher the utility, better your result
  - Always assume your opponent will behave optimally
  - According to the current state, you choose the action that gives you the best achievable utility
  - 
    ```

    - Obviously, time is not enough for you to traverse all possibilities in one game (although in our Tank War, each node only has three child nodes).
    - To deal with this time limitation, you can limit the depth of your Game Tree. Then for the leaf nodes of the tree, you need to design an **evaluation function** to calculate their utilities.
    - This picture above is taken from **CS188**, one of the most famous courses in UC Berkeley. **You can check the original lecture notes if you want to learn more about Game Trees.**
- **Finite State Machine**
    - You can set up some tasks that a tank should do in a certain state. You can let your tank behave differently in different states so that it could have different effects.
    - Actually I'm not familiar with this approach... You can search on Internet for more details.

Grading

- Task 1 worths 140 points.
- The OJ has a scoreboard for each player, each player will be assigned up to 160 points according to his/her rank.
    - If your AI can beat the most straightforward AI, you can get 60 points,
    - Beat other competitors to get the final 100 points!

Bonus 😺

- Find and report some bugs about the OJ platform.
- Help to improve the OJ platform, we need some experts in `Node.js` 😚

Potential Deduction / HC 😾

- Try to attack/hack the OJ platform without informing the TAs.
- Deliberately submit a program that will paralyze the OJ platform.
- **Deliberately login other student's account.**

Reference

- VG101 FA2020 LAB6 - TankWar, Wang Kaibin
- SU21 CS188 Lecture 4 -- Game Trees I.pdf

End of Story...

Usually reference is the last part of a lab manual, but considering that this is the last time I have a chance to force you to read my sentimental nonsense, maybe it's better to write something... And actually I have a lot to say.

OK, perhaps it's too early, as you are just going to take your first course in C++, and still have two labs to complete. Good luck on your journey with C++, and in case I don't see you: Good afternoon, good evening, and good night.