VG101 SU22 Project 2 - Tank War

July 2022

General Rules

This project is designed to hone your programming skills in C/C++. **It is strongly recommended that you do not leave the entire project to the last minute.** In this project, you and your partner will implement a simplified Tank War game. You may find the main idea of this project is a bit similar to demo project 2, Reversi. However, instead of implementing the given functions, we will provide you maximum degrees of freedom this time. No starter files, no skeleton, you are free to design your own program structure and write everything you like.

This is a group project. Each group consists of two students. In some extreme cases, several groups may be consists of three students. According to what Prof. Zhu mentioned in the first lecture, we have automatically arranged the grouping according to a specified algorithm, aiming to balance the performance of each group. The group division has been released on Canvas. You can find your partner through People - Project2. The deadline for project 2 is **August 5 at 23:59**. No late submission will be accepted.

Grading Policy

- Completeness & Correctness (75%): Please refer to the ***TO-DO List*** below;
- Code Style (10%): The code follows a good style (naming, commenting, indentation);
- Documentation (10%): A **brief** README about your program structure, I/O format, implementation details, etc.;
- Peer Evaluation (5%): Forms to be announced;
- Bonus ($\leq$ 15%): Please refer to the ***TO-DO List*** below.

Introduction

Tank War is a classic computer game. Each player controls a tank in this game, competing to become the last survivor. This time, you are going to implement a simplified two-player Tank War and write an AI to support Player Versus Environment (PVE) and DEMO modes. In this project, you will experience Object-Oriented Programming (OOP) and Layered Architecture Patterns in C++, and experience game programming, and even traditional AI programming and socket programming.

Game Rules

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**.

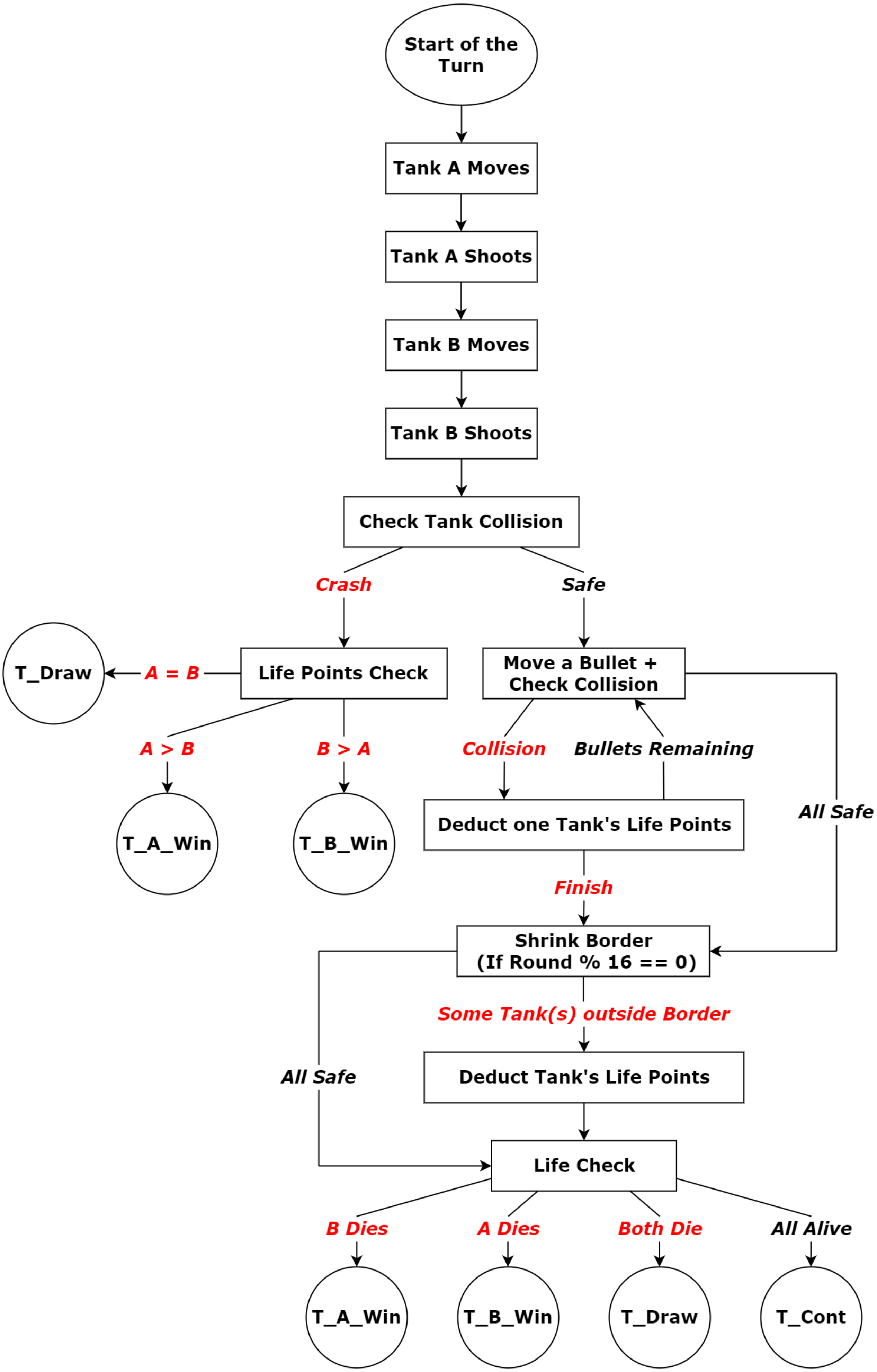
- Tank have 5 **life points**.
- Once a tank is hit by a bullet, 2 **life points** will be deducted.
- To make your life easier, just assume that the bullet moves after the tank moves since this will avoid geometry calculations. Note that each turn there are **three possible cases** where the bullet 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 $20 \times 20 \to 18 \times 18 \to 16 \times 16$...
- 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.

Flow Chart

I know that rules in text are always boring to read and too abstract to understand. 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
T_A_Win
T_B_Win
Deduct one Tank's Life Points
All Safe
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.**

More Vivid Animation

Some tank war veterans (in fact, only me) have found relics of last year's battle. Even though the IT department has cleared the battlefield, some of it remains. You can better understand how to process a tank war game from these relics.

Relics: http://vg101.sometimesnaive.xyz/matches/

Tasks

Your final product is a Tank War game that accepts command-line options, has a command-line interface and supports **PVP (Player vs. Player), PVE (Player vs. Environment/AI) and DEMO (AI vs. AI)** modes. In this part, we provide a general guideline on project goals, sample I/O format and project structure.

TL;DR: TO-DO List

Just in case you get confused by the following project description and don't know what features will ultimately need to be implemented, here's a brief to-do list (and the score for each).

Compulsory Features: 75%

- **Correctly handle all four command-line options: 10%**
  - `-h|--help`: 1%
  - `--log-file`: 5%
  - `-m m|--mode=m`: 2%
  - `-p p|--initial-life=p`: 2%
- **Correctly implement the PVP mode: 40%**
  - Reasonable Command-Line Interface and I/O format: 10%
    - Similar to *demo project 2, milestone 3*, a minimal example is given in *Sample I/O Format* below
  - Correctly process the game: 25%
    - Generally follow the *Flow Chart* above
  - Clearly output the map after each turn: 5%
- **Correctly implement the PVE mode: 10%**
- **Correctly implement the DEMO mode: 10%**
- **Proper instruction messages: 5%**
  - e.g. "You have chosen the PVE mode", "Please input the initial position of tank A", etc.

Bonus Features (See Bonus section and Appendix for detail): Up to 15%

- Implement an advanced AI that beats the baseline: 5%
- Develop a Graphic User Interface: 5%
- Support cross-host PVP mode: 5%

Code Style: 10%

We don't have strict code style requirements like VE280 (i.e. maximum lines per function, number of sub-functions, clang-tidy checks, etc.). Name each variable, function and class clearly, write brief comments for each function and class, and follow the Object-Oriented Paradigm and Layered Architecture Pattern (or your preferred approach). This should be enough to pass this project's code style check.

Documentation: 10%

Don't write too long. My most hated and worst performing courses in JI are VP141 and VP241.

We provide some sample questions to help you better construct your README. You do not need to follow these questions strictly, but generally, you should write your README around these questions.

- **(Important)** How to compile and run your program?
- Did you use the Layered Architecture Pattern? If yes, briefly describe the components of each layer. If not, explain your pattern;
- Did you use the Object-Oriented Paradigm? If yes, briefly describe the `classes` you defined and their meanings. If not, state your paradigm;
- Your program's I/O format including the log file;
- (Optional) Some typical issues and bugs you encountered in the development, and your solutions;
- Did you implement any bonus features? If yes, state the feature and your approach.

Peer Evaluation: 5%

Official forms to be announced. It should be similar to the table below. **Do not maliciously evaluate your partner!**

Name	Score (0-5)	Tasks
Haorong 1	5	Sleeping
Haorong 2	5	Coasting

Project Goals

You start by thinking of the big picture and what you want your program to do. You have tanks and need to support PVP, PVE, and DEMO modes, so defining options specifying the game mode and initial tank life points make sense. Beyond those basic arguments, you would also like your program to define a log file saving all the details of a game. Of course, you do not forget the usual help menu explaining how to run the program and use the options. Noe that both abbreviated and long options must be supported, e.g., help should be displayed when either of the `-h` or `--help` command-line options are provided.

```
user@ubuntu:~$ ./tankwar -h
  -h|--help              print this help message
  --log-file filename    write the logs in filename (default: tankwar.log)
  -m m|--mode=m          play tank war in m (PVP/PVE/DEMO) mode (default: PVP)
  -p p|--initial-life=p  each tank has p life points initially, p >= 1 (default: 5)
```

Now that you have clearly defined the main lines of the project. At this stage, the only thing left for consideration is the I/O format, including the log file format, which is similar to what the DEMO mode should be displayed. DEMO mode is based on the regular PVP mode, but two clever AIs replace two foolish human players.

From the tank war battlefield relics, you have found a copy of the sample I/O format at that time. Although very crude, you can use it as a reference.

Sample I/O Format (Minimal Example)

This part gives a sample I/O of project 2. **Please note that this is just a minimal I/O example. As a user-friendly program, your I/O format should be more decorated and detailed.** For example, you may have some kind instructions and print the current game board after each turn, just like what you did / will do in demo project 2.

Sample command-line options:

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

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
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
```

Corresponding outputs in `stdout`:

```
Tank A Wins
```

Sample output of map:

```
A: 5, B: 1, Turn: 36
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-|B| | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-|*| | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-|A| | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-| | | | | | | | | | | | | | | | |-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
```

In the command line options, `--mode=PVP` indicates that you start the PVP mode, namely, both tanks' initial positions and following actions are taken from user input. `--initial-life=5` indicates that the initial life point of each tank is 5.

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. Here the directions and actions are defined as below (feel free to change this definition in your program),

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

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

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.**

On the map, `"A"` and `"B"` represent tanks A and B, `"-"` represents the region outside the border, and `" "` represents the region inside the border. We also output the current turn and two tanks' life points above the map.

Once you know what they expect to complete and how they could extend their game in the future, you move on to the program structure. You design it to specifically fulfill all project goals.

Program Structure

One of the fundamental design patterns when developing an object-oriented project is **Layered Architecture Pattern**. This pattern partitions all of an application's classes and interfaces (future and current) into layers. It brings much flexibility while also saving much rewriting when adjusting the code and allowing faster debugging in case of a problem. The idea is to organize the code in terms of layers and prevent any function for a lower layer from calling functions from a higher layer. Functions from a higher layer can use functions in the same layer or a layer below.

In the case of Tank War, you and your partner identify three main layers from lowest to highest:

Layer 1

`classes` needed for the well functioning of the game:

- `class` members definitions, e.g. tanks and bullets;
- Corresponding `class` methods to handle the data structures.

Layer 2

`classes` needed to play the game:

- e.g. A `class Game` with methods to:
  - Initialize a game;
  - Move tanks and bullets in each turn;
  - Judge and calculate and the end of each turn;
  - Print current game state.
- e.g. A `class AiBrain` with methods to:
  - Initialize an AI Brain;
  - Decide next action according to current game state.

Layer 3

Functions needed for the user to interact with the computer and play:

- A function to display the game's current status, e.g. board, tanks' life points, turn, etc.;
- A function reading user inputs and process them;
- PVE and DEMO modes, replace one or two human players with your AI.

Remember that the goal is to save time and render the program clearer. So now imagine that functions in Layer 1 could call functions in Layer 3. Then let's say a function from Layer 3 that used to take two `int` as input now takes one `double` and one `int`. Then any function that uses it must be rewritten, i.e. changes in Layer 3 might impact functions in Layer 1. This would be very bad since, for instance, changing how a player interacts with the computer would mean redefining low-level data structures also used in Layer 2. In other words, the whole program would need to be rewritten, just for a simple adjustment!

Besides, since the third layer is the User Interface (UI), having it as the top layer, separated from the rest of the program, allows the programmer to write different kinds of UI. For instance, a programmer could start with a Command-Line Interface but then decide to implement a Graphical User Interface (not required in this project). With layer programming, the UI does not impact the rest of the program, so adding new types is very simple. Similarly, once all the lower-level functions are ready, PVE and DEMO modes are simple, with the same functions for real users, but called with random parameters or specific AI strategy.

Since it's just a general draft, your layers are to be used as rough guidelines, and some minor adjustments might be needed, and new functions or data structures added. It is possible to add more layers as long as no lower layer needs a higher one.

Bonus

Warning: A group not fully completing all the compulsory tasks **will not receive any bonus**. So please complete the required part first. To prevent peer pressure, bonus points are not proportional to the time and effort it takes you to implement these features. Make sure you and your partner have enough time and interest before starting to develop bonus features.

Optional tasks bringing a reward:

- Write an advanced AI using approaches like minimax (Appendix A, or even more powerful);
- Use a toolkit to implement a GUI, e.g. GTK, Qt;
- Create a real game setup that can be played over a network (Appendix B).

Project Submission

Before submitting the project on Canvas, **ensure the project compiles on JOJ**. The JOJ compilation test will be opened soon.

- A project not compiling with the following flags will not be graded: `-O2 -Werror -Wall -Wextra -Wpedantic -std=c++1z`;
- A project submission which directly crashes when run will not be graded;
- JOJ can be used to ensure the written code compiles and complies with the C++ standard. If using a `Makefile` or `CMakeLists.txt` file with JOJ, ensure the above flags are enabled. **It recommended to write your own `Makefile/CMakeLists.txt`;**

- If the submission uses external libraries, e.g. GTK or sockets, please inform the teaching team when uploading the code, and briefly conclude your design and implementation in your README.

FAQ

This section lists Frequently Asked Questions (FAQ).

- I have no idea where to start and what to do.
    - Log on to Piazza and discuss with other students and the teaching team. Clearly explain what you do not understand and why you feel stuck, **do not ask for a solution**. If several opinions appear to be valid, determine which ones are the best and most reasonable. Document your choices in the README file. Feel free to edit or refine others' questions and answers.
- I am very busy with the project and do not have time to work on the assignments.
    - Change your work strategy: **first solve the assignments and then move on to the project**. Several exercises from the assignments can be partially reused in the project. Directly starting with a challenging task is a waste of time. Assignments are designed to help you progress, and milestones have been organized with the assignments in mind.
- Is there any easy and clean way to parse command line arguments?
    - Look at the header file `getopt.h`
- How should I provide the location of the logfile?
    - A file location can be expressed using either a **relative or an absolute path**. As the absolute path is "computer-specific," it is not a good idea to define an absolute path in the program. Therefore in this project, only a relative path should be used inside the program. A user should however be able to use either an absolute or relative path when providing a file location as a command-line argument.

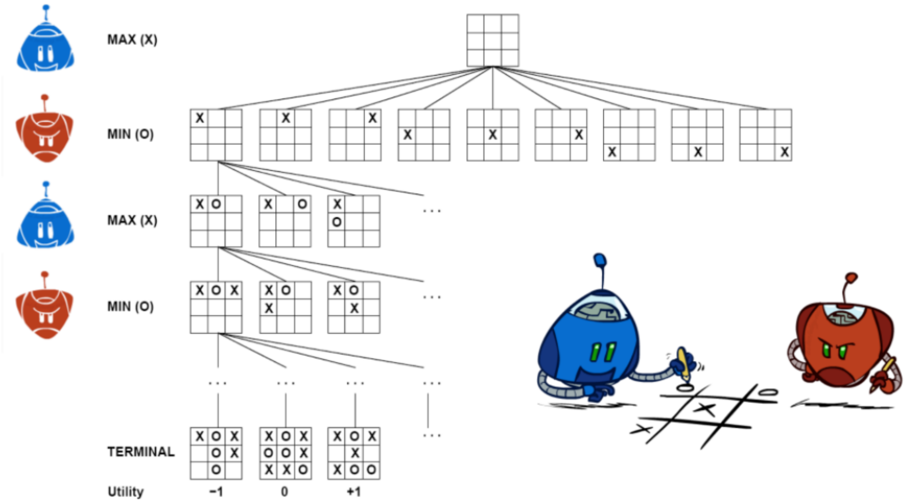
Appendix A

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

Tic-Tac-Toe Game Tree



- 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.

Appendix B

Develop a REAL game!

In real life, the game server and client are separated into two programs and run on different computers. They can communicate using some protocols that need to be precisely specified. After completing all the other tasks, you may want to work on that.

There is no restriction on the strategy allowing the separated server and clients to communicate. The minimum requirement to obtain a bonus is the ability of a server to run separately from the clients while being able to communicate with them. A larger bonus will reward work that can run over a network.

You found the following information which you decided to use as a reference without necessarily precisely following it. In particular, you notice that to allow their work to be compatible with yours, you need to follow a specific API that will ensure the compatibility between all the servers and clients regardless of their authors.

Socket Communication

In practice, a socket usually refers to a socket in an Internet Protocol (IP) network, where a socket may be called an Internet socket. This is, for instance, how the Transmission Control Protocol (TCP), a protocol for one-to-one connections, works. In this context, sockets are assumed to be associated with a specific socket address, namely the IP address and a port number on the local node. Similarly, a socket address is also defined on the remote node so the remote process can reach its associated socket. Associating a socket with a socket address is called binding. Once an address has been bound to a socket, the program that created it starts to receive, i.e, listen, messages sent to that address.

API (Application Programming Interface)

All traditional mails have stamps, addresses, and zip codes. All emails have email addresses (sender, receiver, and cc), subject and content. To communicate, two sides need to have a standard communication format, which is usually called communication protocol in Computer Science. Similarly, to communicate between servers and clients, they should agree on a series of criteria, often called Application Programming Interface (API).

Reference

- **VG101 FA2020 Lab 6 - Tank War, Kaibin Wang**
- **VG101 FA2020 Project 2, Manuel**
- VG101 FA2021 Lab 7 - Tank War, Haorong Lu
- Berkeley CS188 Lecture 4 -- Game Trees I.pdf