Tank World
A framework for AI competing

Weiguang Zhou 100136054 [*]

December 9, 2015

1 Requirements

1.1 Introduction

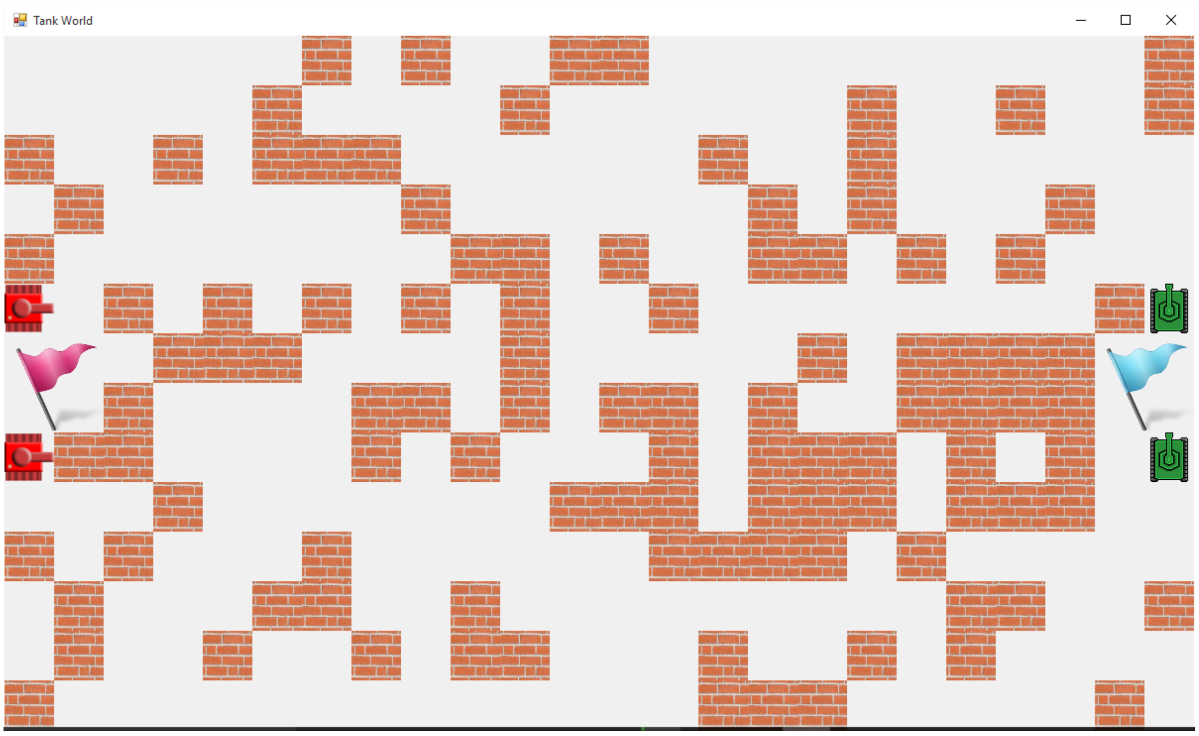

Figure 1: Game GUI

[*]Student in School of Computering Sciences

Tank World is an interesting Computer-vs-Computer emulator, in which the red team and the blue one try their best to capture the flag of enemy or terminate every tank of the other team to win the game. It is used as a framework to load two algorithms and test which one is better.

1.2 Map

1.2.1 Blocks

The Map is a rectangle with m*n square Blocks. A Block is either passable or not.

1.2.2 Map Generation

The Map is automatically generated. The map must be passable. Every tank can go to the enemy's head-quarter.

1.3 Tanks

Tanks can move from one passable block to any neighbouring passable block. Tanks can fire and destroy other tanks.

1.4 Games

1.4.1 Goal of the Game

To win the game, a team either terminates every tank of the other team or captures the flag of the other team.

1.4.2 A round

When the two teams fight, they take actions in turn. This is very like two people having a board game match. In one round, two sides execute the following actions as listed in table.

Sequence	Chess Piece	Controlled By
1	Red Tank A	Red Player
2	Blue Tank A	Blue Player
3	Red Tank B	Red Player
4	Blue Tank B	Blue Player

1.4.3 An Action

An Action is one of the following:

Turn the tank changes direction it faces. This includes turning backward.

Move the tank moves forward. It moves one block a time.

Fire the tank fires forward.

1.5 External AI

The two teams are controlled by external AIs. The program run the game thousands of times to find out which AI is the best. The AI can access the map and positions of other tanks. The AI takes actions based on the current situation.

2 Technical Solution and Challenges

2.1 Technical Summary

Programming Languages	C# 5.0, F# 5.0 and VB.net 5.0, Octave
.Net Version	4.5
IDE	Visual Studio 2015, Octave 4.0.0
Source Code Management	Team Foundation Server
Graphics Libraries	OpenTK
Algorithms	Dijkstra Short Path, Union Find
Machine Learning Algorithms	Logistic Regression, Neural Network
Third Part Code & Libraries	Union Find(Java), OpenTK(DLL)

2.2 Algorithm and data structure

Algorithm is used when:
Checking if the map is passable, meaning if a tank is able to go to the enemy's flag.(union-find, percolation). The UnionFind class is from PrincetonRobert Sedgewick (2011)
Calculating the shortest path from one place to another on the map.(Dijkstra's algorithm)(see)

2.3 Graphics

This project uses OpenTK. OpenTK is a .net wrapper of OpenGL. It is free and open-sourced. (Fiddler, 2008)

2.4 Multi-Thread or Parallel Implementation

Multi-threads is important when

- Thousands of games are being run at the same time. Or to say, the two players are matching in batch mode.
- When the result of batch match is being displayed. Without multi-thread, the UI will freeze and the user experience is very poor. Thus, we need them run at the back-end without affecting the UI. Further more, we also need the result to

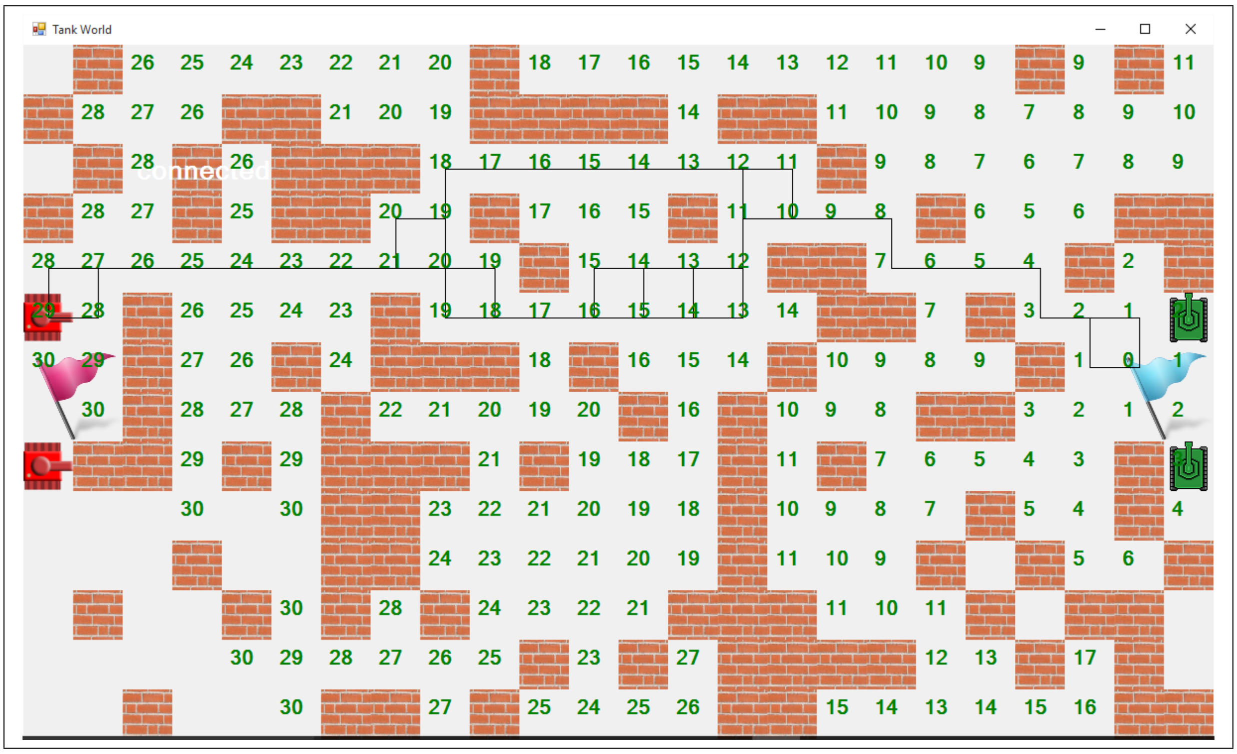

Figure 2: The shortest path

be displayed instantly. It is just like in the election, websites update the voting instantly without refreshing the web page.

Traditionally, or at least in C++, Java and C# 1.0 to 3.5, we have to declare new threads, handle callback, merge threads and etc. However, from C# 4.0, we can use await and async keywords, which have made using multi-thread much easier. Thus in the programme, these two keywords are used.

2.5 Generic and Flexible Architecture

External algorithms are loaded into this game by DLLs. Those DLLs must implement certain interfaces.

2.6 Programming code optimization in terms of speed and memory usage

The Union-Find algorithm and Dijkstra's Algorithm are used to optimise the system.

2.7 Multi-languages and Paradigms

Languages Used: C#, VB, F# and Octave
Object Oriented Programming: C#, VB.net

Functional Programming Paradigm:

C# LINQ, Higher-order functions, Asynchronous functions, Tuples, Curry Functions.

VB.net LINQ, Higher-order functions

F# description

The Octave code is base on the code from Princeton, my code is in three files: Tank_nnCostFunction.m, Tank_predict.m and TankTrain.m.

3 Machine Learning Library

The most difficult part of this project is to implement the machine learning library. Because this project can be viewed as a classification problem in which the players have to decide where the tank will go or whether it should fire.

3.1 Logistic Regrassion

The part of logistic regression was successful. I implemented binary classification and multi-classification all in C# code.

3.2 Neural Network

I trained the parameters or θ in matlab and used θ of C#.

3.3 Decision Tree and AdaBoost

I implemented the entropy class, the stump class. However, the development of AdaBoost stopped because I do not know what to do if individual models do not support weights.

4 Test and TDD

This project has 32 test cases in the projects of TankWorld.MachineLearning.Test and TankWorld. The develpment was driven by them.

A great portion of them are the play tests in which two players fight without GUI. Some players are designed to be more clever than others, so they must win more to let the test case pass.

4.1 Mutual Test

Some algorithms are test each other. That saves developers time to check the result one by one. The Union-Find Algorithm and the Dijkstra Shortest Path Algorithm can check each other. Logically, if point A to B is not passable, it is not in the same union.

The UnionFind_Dijkstra_Test1000 test case run the two algorithms 1000 times and check if their results are contradictory.

5 AI Players

5.1 Wandering Player

A wandering player has the tanks wander around without any specific purpose. Winning is based on luck. Logically, when two of that kind players match, there is a very high probability the match will end in tie. The reason is they just cannot reach the destination within 2000 rounds and the match will run out of time and end.

5.2 Flag Remover

A flag remover goes to the enemy head-quarter directly. In 1000 matches, No 1 wins 286 time, No 2 317 time. The rest are ties. Roughly, No1 and No 2 each win 30% of them games, and 40% of the games end in tie. That is because, the two players have their tanks go to enemy head quarter directly and the distance from a red tank to the blue head quarter is similar to the distance of a blue tank to the red head quarter.

However, when a flag remover matches with a wandering player, the later has a slim chance to win. In 1000 match between a flag remover and a wandering player, the flag remover or the red player won 954 times. In contrast, the wandering player or the blue player only won 3 times. The flag remover overwhelmed the wandering player. In other word, the algorithm of the flag remover is much better than the algorithm of the wandering player.

5.3 Programmed Player

The programmed player is explicitly programmed. The player knows how to avoid being attacked by others. When there is no danger, the tanks goes to enemy head quarter. Otherwise, the tanks try to terminate enemies. The logic is demonstrated by the figure 3. According to the running result, the programmed player defeats the previous two players. The following table demonstrates the results.

Player1	Win	Player2	Win	Tie
Programmed Player	44	Programmed Player	42	14
Programmed Player	93	Flag Remover	7	0
Programmed Player	96	wandering Player	4	0

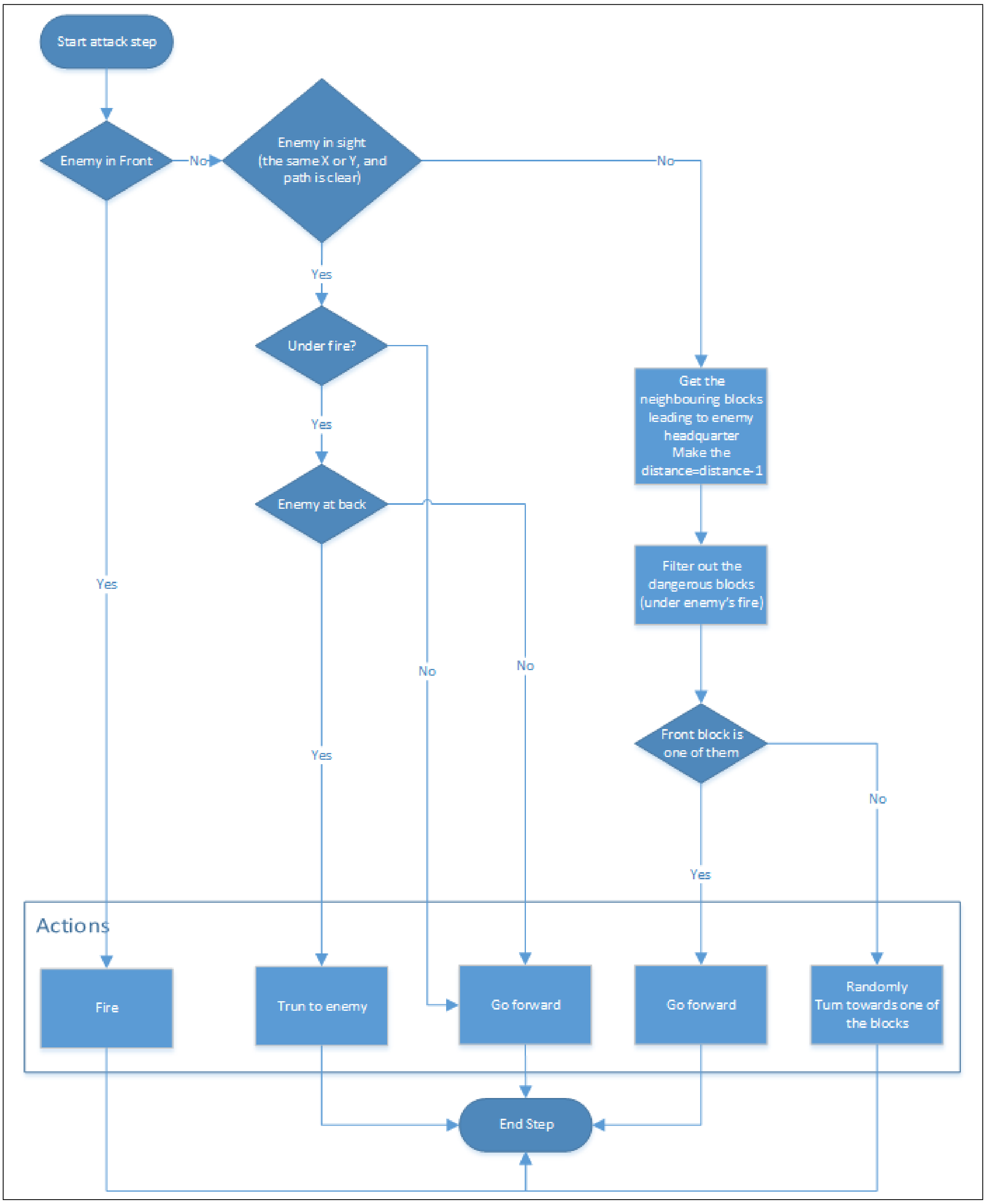
Start attack step
Enemy in Front
No
Enemy in sight (the same X or Y, and path is clear)
No
Yes
Under fire?
Yes
Enemy at back
Yes
No
No
Yes
Get the neighbouring blocks leading to enemy headquarter Make the distance=distance-1
Filter out the dangerous blocks (under enemy's fire)
Front block is one of them
Yes
No
Actions
Fire
Trun to enemy
Go forward
Go forward
Randomly Turn towards one of the blocks
End Step

Figure 3: Programmed Player

5.4 Machine Learning Players

5.4.1 Logisic Regression

"Arthur Samuel in 1959 defined machine learning as the'field of study that gives computers the ability to learn without being explicitly programmed'." (Munoz, 2014)

Given in every step, the decisions of player are limited. The player has to choose one of the following.

- Move Forward
- Turn Left
- Turn Right
- Turn Back
- Fire

Making a decision to choose from the above 5 options is typically a classification problem. The result of the decision can be represented by a vector $y \in R^5$. The following table demonstrates the mapping between tank actions and vector values:

Action	Vector
Fire	[1, 0, 0, 0, 0, 0]
Move Forward	[0, 1, 0, 0, 0, 0]
Turn Up	[0, 0, 1, 0, 0, 0]
Turn Right	[0, 0, 0, 1, 0, 0]
Turn Down	[0, 0, 0, 0, 1, 0]
Turn Left	[0, 0, 0, 0, 0, 1]

In order to calculate the y vector, a group of features are required, they are defined as vector $x \in R^{40}$.

Some Features

x	Feature	Math Representation
x_{13}	Red Tank 1 faces up	Yes-1, No-0
x_{14}	Red Tank 1 faces right	Yes-1, No-0
x_{15}	Red Tank 1 faces left	Yes-1, No-0
x_{16}	Red Tank 1 faces down	Yes-1, No-0
x_{17}	Red Tank 1 X(coordinate)	Scaled, X/Width-1, So the value is between 0 and 1.
x_{18}	Red Tank 1 Y(coordinate)	Scaled, Y/Height-1, So the value is between 0 and 1.
x_{19}	Red Tank 2 faces up	Yes-1, No-0
x_{20}	Red Tank 2 faces right	Yes-1, No-0

Now we have n=40 features, k=5 options. Suppose we have m samples, and m is a positive integer. We now have $X \in R^{\mathrm{m*n}}, Y \in R^{\mathrm{m*k}}$. The relation between features X and results Y is defined as Y=f(X). The hypothesis of that is H(X). We now have to find a function H that fits f, meaning the error is minimised.

To resolve this problem, the first machine learning model would be a simple logistic regression model, or a simple classifier. The hypothesis is

$$h(x) = g(\theta_0 + \theta_1 * x),\ \mathrm{g(z)} = \frac{1}{1-e^{-z}}\ .$$

The square error is $\sum_{i=1}^{n}(h(x_{\mathrm{i}}) - y_{\mathrm{i}})^2$. The gradient descent function is

$$\theta_{\mathrm{j}} = \theta_{\mathrm{j}} - \alpha\frac{1}{m}\sum_1^m(h(x^{(\mathrm{i})}) - y^{(\mathrm{i})})$$

Consider the overfit problem, the regularised function is:

$$\theta_0 = \theta_0 - \alpha\frac{1}{m}\sum_1^m(h(x^{(\mathrm{i})}) - y^{(\mathrm{i})})$$
$$\theta_{\mathrm{j}} = \theta_{\mathrm{j}} - \alpha(\frac{1}{m}\sum_1^m(h(x^{(\mathrm{i})}) - y^{(\mathrm{i})}) + \frac{\lambda}{m}\theta_{\mathrm{j}})(j \neq 0)$$

5.4.2 Evaluation

The number of samples or m, α, λ need to be evaluated. The candidates of these are as follow:

Variabe	Candidates
m	1000, 2000, 3000...10000
α	0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30
λ	0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30

The system need to try all of the above values, choose one of each to balance between bias and variance, precise and computational expense.

5.4.3 Samples Generation

Ideally, the samples should be generated by human beings who are good at the game. The system records every step he or she makes and restore them as samples. However, for the time being, we do not have time and resources to do so. Thus the possible way is to record steps made by the programmed player and use them as training samples.

5.4.4 Fighting Performance

According to the running result, the programmed player defeats the previous two players. The following table demonstrates the results.

Player1	Win	Player2	Win	Tie
Logistic Regression	10	Logistic Regression	9	81
Logistic Regression	18	Programmed Player	25	57
Logistic Regression	67	Flag Remover	33	0
Logistic Regression	56	wandering Player	36	8

5.4.5 Neural Network

The neural network player is very similar to the logistic regression player. They shared the training, cross validation and test samples. In this case, the neural network has an output layer with 6 neurons and an input layer with 40 neurons. I have inserted a hidden layer with 15 neurons. The parameter or θ was trained in Octave and used in C#. The performance is as follow.

Player1	Win	Player2	Win	Tie
Neural Network	2	Neural Network	1	97
Neural Network	3	Logistic Regression	17	80
Neural Network	5	Programmed Player	43	52
Neural Network	2	Flag Remover	91	7
Neural Network	51	wandering Player	49	0

5.4.6 Problems

Both the logistic regression player and the neural network player are not smarter than the programmed player. That is not a technical problem. Ideally, the logistic regression player should defeat the programmed player and the neural network player should defeat the logistic regression player. However, both of them are apprentices of the programmed player. They learnt from the programmed player.

The way that the machine learning players defeat the programmed player is to learn from a human being. We let a human player or two play the game and record every step.

6 Technical Guidance of Developing External Players

The external player is embodied by a DLL that depends on TankWorld.Core DLL and used by TankWorld.GameManager DLL. See Figure 4

The major types and their dependencies are demonstrated in figure 5. For detailed information of every class and every function, please check the comments of them.

The types a developer should pay attention to are *IPlayer* , *BasePlayer* and *PlayerAttribute.* An external player class must implement *IPlayer* Interface directly or indirectly. In the indirect way is to inherit *BasePlayer* Class which implements *IPlayer* Interface. The external player class must be decorated by *PlayerAttribute* Attribute and the *DisplayName* Property must be given. This is used by the UI to distinguish between external players.

The player classes will be used by *GameManager Class.* The GameManager class will organise the match between two players and be the judge of the game. The GameManager Class firstly creates the map. It then creates two players from internal or external player classes. The player instances will be given several parameters including the opponent or the other player, the map, two tanks on the map. The enemy tanks can be accessed by the opponent property. Each item on the map can be accessed via the map instance.

The external player DLLs are put into the *plugin* folder as well as other related files, so please point your files in the plugin folder.

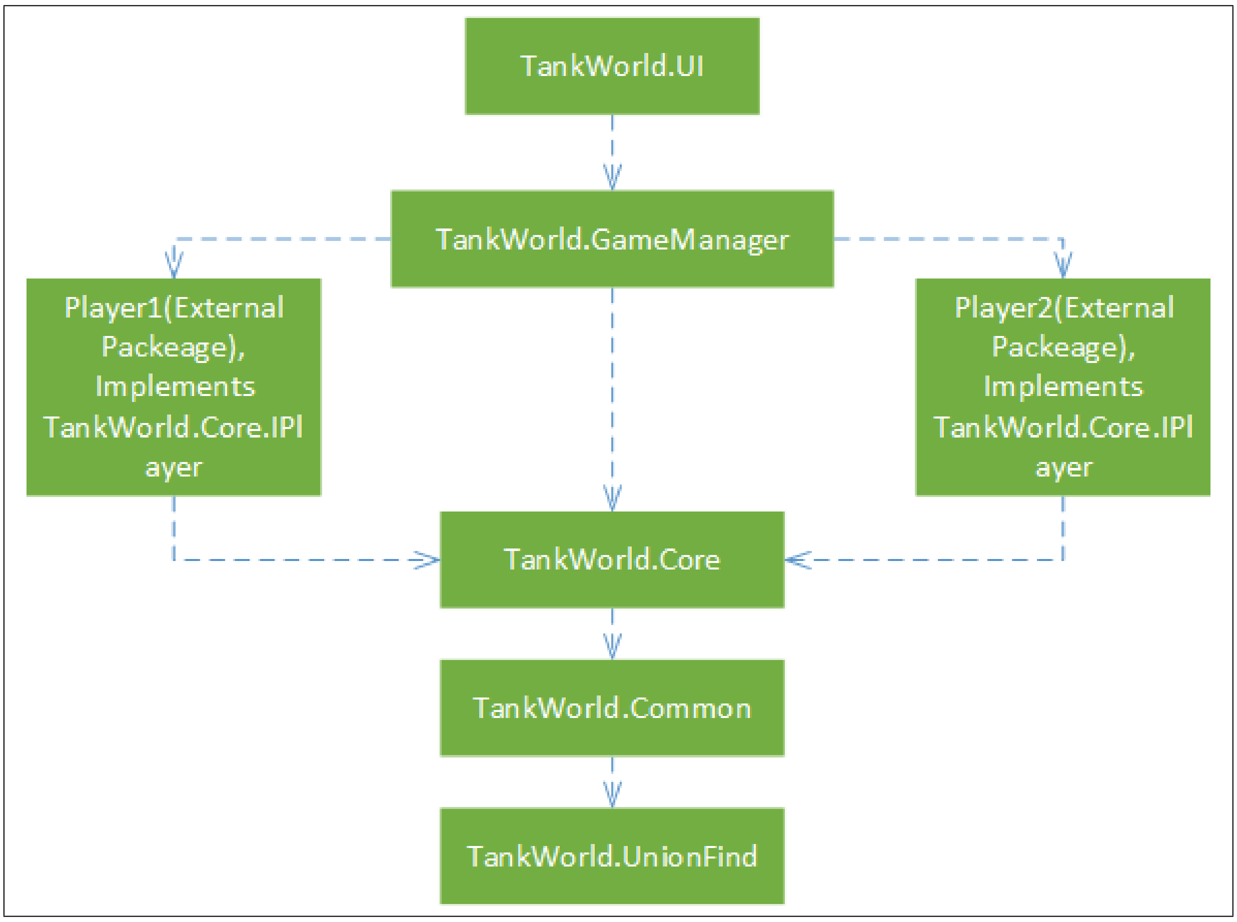
TankWorld.UI
TankWorld.GameManager
Player1(External Packeage), Implements TankWorld.Core.IPlayer
Player2(External Packeage), Implements TankWorld.Core.IPlayer
TankWorld.Core
TankWorld.Common
TankWorld.UnionFind

Figure 4: External Player

6.1 Conventions

A player class name must ends with -Player. Although it does not affect the execution, a meaningful name is very helpful.

7 Roles and Assignment

Only one person. Not applicable.

References

Fiddler (2008). The open toolkit library.

Munoz, A. (2014). Machine learning and optimization.

Robert Sedgewick, K. W. (2011). *Algorithms, 4th edition*. Addison-Wesley Professional.

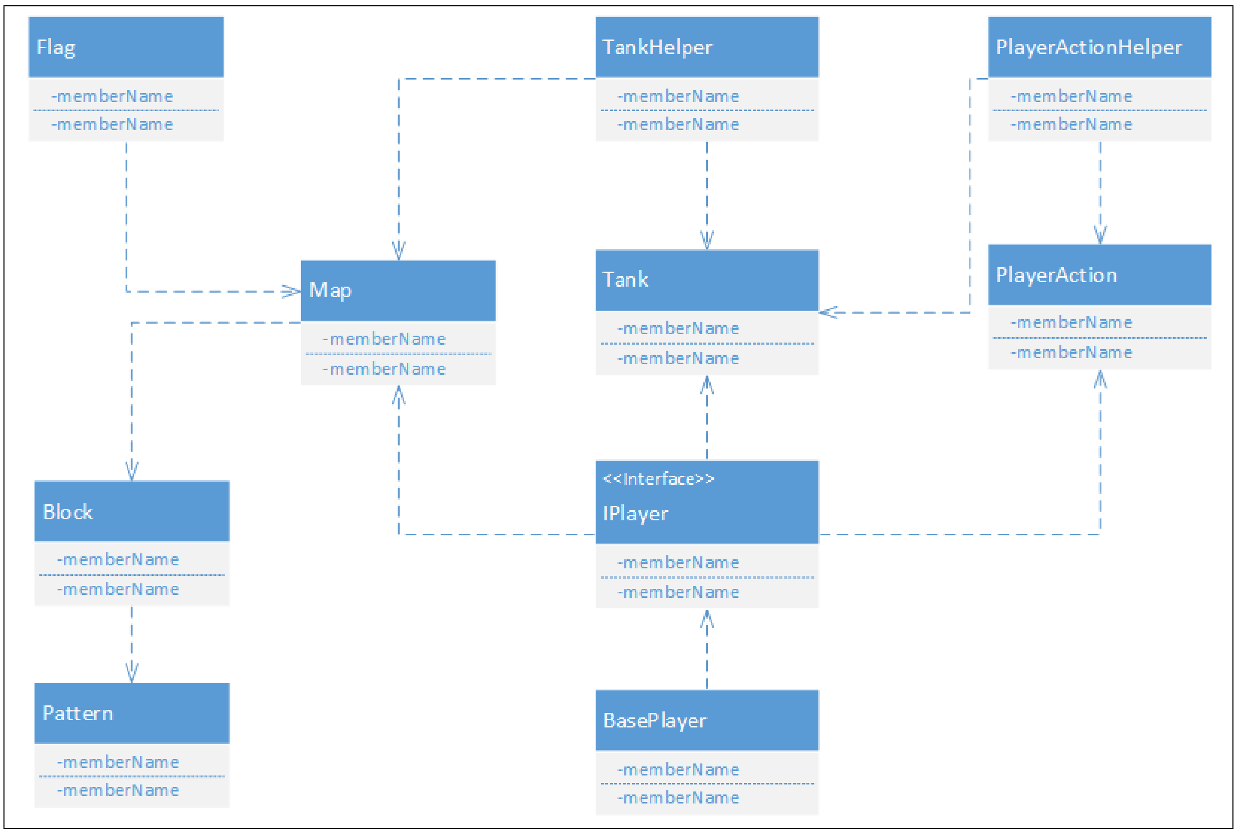

Figure 5: Core Types

8 Evidence of IID or AP

The document per si is an evidence of IID. In this document, there are some game pictures generated by the GDI+ UI which is no longer used due to its flashing animation. I changed it to OpenTK eventually.

8.1 Document First

The design phase was done by UML and Visio. While I was writing the code and found something wrong with the design, I went back to the UML, modified it before I continued. Some UMLs are given in this document.