EEE102 Assessment 2 Report

Yimian Liu 1717608

April 8, 2019

1. Introduction

In the previous lectures, we obtained a basic understanding on C++ and had for the first time programed with several objects and classes. In the last assignment, a fraction class which contains several methods and properties had been developed. We had achieved that using the fraction class to declare objects and then this objects to operate just like others standard class such as int and string. Then in this assignment, we are required to further developed an iFraction class which should be a sub-class of Fraction and also to design a function which provides the method to convert Fraction to iFraction. Besides, we are also suggested to modify and developed a game basing on the code provided.

This report will detailly classify and present how this assignment be conducted with the instruction of Software Development Process (SDP).

2. Problem Statement

2.1 iFraction Class

This exercise requires to design a sub-class of the former Fraction class and it should be operated as what a mixed fraction should do. Which means that, just as the Fraction class, the iFraction class should also have functions such as be overloaded in expression, construct, destruct etc., which can be classified into input, output and operation three sections.

2.1.1 Input

Basic Requirement

- The iFraction can be declared with nothing and means 0. E.g. Fraction a;
- The iFraction can be declared with a integer, a numerator and denominator. E.g. Fraction b(3, 3,-4);
- The iFraction can be declared with simple numerator. E.g. Fraction c(5);

Advanced Requirement

- The iFraction can be declared with another iFraction. E.g. iFraction f = b;
- The iFraction can be declared with another Fraction. E.g. iFraction f = c;
- Input is normalized and simplified.
- Only the integer or numerator can be negative.
- Decimals should be converted to iFraction form.

My Requirement

- The iFraction can be declared with a double type number. E.g. iFraction d(-3.14);
- The iFraction can be declared with double number directly. E.g. iFraction e = 1.3333;
- Recognize repeated decimals and transfer it to corresponding iFraction.
- Can be input use cin with decimals. E.g. cin >> f; (input: 3.14)

2.1.2 Output

Basic Requirement

- There has an output.

Advanced Requirement

- Can output decimals. E.g. f.val();
- Can output integer, numerator and denominator. E.g. f.integer(), f.itop(), f.bottom();

My Requirement

- Can directly use cout to output.

2.1.3 Operation

Basic Requirement

- Add, subtract, multiple and divide. E.g. (+, -, *, /)
- Compare base on values. E.g. (==, <, <=, >, >=, !=).

My Requirement

- Get opposite number and reciprocal. E.g. Fraction f(1,3,4); -f = -1(3/4); ~f = 4/7;
- Get remainder. E.g. f%2;
- Allow f++ and ++f, f--and --f.
- Allow f +=?, f -=?, f *=?, f /=?.
- Can directly operate with other numbers.

2.2 Game

This program requires us to change the codes which are marked as ??????? and let the code can run on the computer firstly. Then, CRC card are needed to be generated for each class. Figure out the relationship and hierarchy of these class and their labeled members and then draw them into a hierarchy chart.

After finishing the first, we can move on to designed other two class which describe the methods and properties of archer and mage referring to the code of swordsman. Then, classify them in to hierarchy chart just as what had down for the other classed. Modify the main function to make the role of opponents can be randomly selected from all these three role classes. Finally, try to add some luck part to the game.

3. Analysis

3.1 iFraction

3.2.1 Variables

As this is a subclass of Fraction, all of the three variables such as _top, _bottom and _isNegative are designed to be fully used. Which means that, we still use the data struct of Fraction class to store and operate data.

3.2.2 Input & Output

Every thing is just like the father class, nearly all of the methods are inherited from the Fraction class. The only thing that iFraction need to do is to convert the input to what the father class can config and memory them in the property of father class. When output, convert the result from father class's property to mixed fraction format.

To be specific, here are mainly four output formats. The first one is directly use iFraction in a cout stream like this cout << iFraction. This will push a string to cout in format like '-1(1/3)'. This will return a string just like it shows in cout stream. The third way is use iFraction.val(). This will return a double decimal. The fourth way is to use iFraction.interget(), iFraction.itop() and iFraction.bottom() to get its integer, numerator and denominator.

3.2 Games

To solve these five problems which were stated in the task sheet, we need to understand the original code firstly. Then, which the knowledge of precompile, dynamic memory, pointer etc., try to fill all the six questions spread over these code files. After doing this, we can further understand these codes and focus on the relationship of these classes and functions and classify them to a hierarchy chart. In this process, it should be careful regarding the action scope of each method/function and variable/property.

After grasping a deep scope of this game structure, we then can try to imitate the swordsman class and write another two class to achieve the roles of archer and mage. Besides, modify the main function and let all these threes roles can be presented in the game. Finally, we can also add some luck part to the game by add some random in math library to where can let player feel lucky such as the attack decreased HP.

4. Design

4.1 Class Inherit

Inherit is a very interesting idea in class that a subclass can partly inherit the methods and properties from father class and use them as it is one of its own functions or variables. By using this character, we can quickly develop an iFraction sub class basing on a powerful Fraction Class.

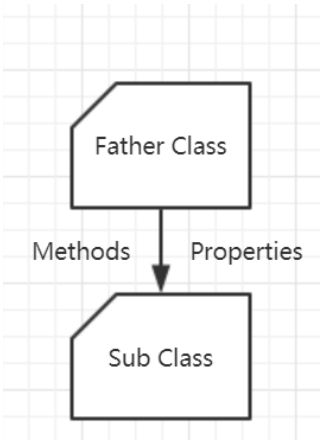


Figure 1 – Inherit

4.2 CRC Card of container

From my understanding, a CRC card is tool which can help developers to classify their ideas and let them collaborate better when utilizing an object-oriented language such as C++. To further describe this method, giving every class a table and list its classified and simplified method/responsibility in the left side and fill its relationships/dependency in the right side. Following this instruction, the CRC card of container had been made.

container	
set the items numbers - public	
get the number of heal - public	
get the number of magic waters - public	
display the items - public	
use heal - public	
use magic water - public	

Table 1 – container CRC card

As it shown in the above table, the container class has six methods, which achieves all of the functions that a bag in a game should have. As it is an individual class and has no dependency, the left side was filled with nothing.

4.3 CRC Card of player

Player(abstract)	
normal attack - public	Showinfo()
special attack - public	swordsman
level up judgement - public	archer
AI for robot - public	mage
character's HP and MP resume - public	container
report whether character is dead - public	
check whether character is dead - public	
consume heal, irrelevant to job - public	
consume magic water, irrelevant to job - public	
display character's job - public	
possess opponent's items after victory - public	

Table 2 – player CRC card

Similarly, the CRC card of player are shown as the above table. Different form the container, it has 11 methods and also five dependency. In these methods, the first four are pure virtual functions which can only be achieved in the corresponded sub classes, which also make the player class being a abstract class, meaning that it cannot be instantiated independently. Also, it has a friendly relationship with the showinfo function, which allows this function to have a higher level authority to operate its method and property.

4.4 CRC Card of swordsman

swordsman	
normal attack(on AP, DP) - public	Player
special attack - public	
level up judgement - public	
AI for robot - public	

Table 3 – swordsman CRC card

This is one for the required sub class of player class, aiming to further describe a role call swordsman and make it can be realized when used. It has mainly four methods, to achieve the required and characteristic functions of this certain role. Of cause, this class need the dependency of the player class.

4.5 CRC Card of archer

archer	
normal attack(on Speed, DP) - public	Player
special attack - public	
level up judgement - public	
AI for robot - public	

Table 4 – archer CRC card

Similarly with the swordsman, this is another role class inherit from the player class.

4.6 CRC Card of mage

mage	
normal attack(on EXP, DP) - public	Player
special attack - public	
level up judgement - public	
AI for robot - public	

Table 5 – mage CRC card

This is another class to define the role of mage.

4.7 Hierarchy chart

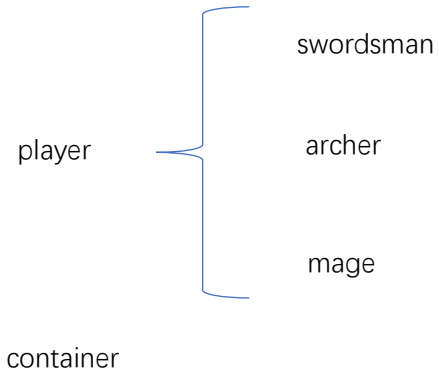


Figure 2 – Hierarchy chart

As it shown in Figure 2, a clear hierarchy relationship can be figured out. Just as it indicated, the subclass swordsman, archer, mage are parallel with each other and inherit from player class. While, player is a friendly class with container class. This allowed player can get the information and interact with container class.

5. Implementation

There are four Cpp file of this assessment. Since I have written some codes that belong to C11, it is recommended to use a complier such as gcc.

ex1/Fraction.h
This file includes the source code of Fraction Class which belong to the second exercise. It is also support online view from the following url.
url: https://github.com/string1995/eee102/blob/master/as2/ex1/Fraction.h

ex1/ex1.cpp
This file contains the test code of iFraction Class.
url: https://github.com/string1995/eee102/blob/master/as2/ex1/ex1.cpp

ex2/main.cpp
This file contains the main function code of ex2 game.
url: https://github.com/string1995/eee102/blob/master/as2/ex2/main.cpp

ex2/player.cpp
This file contains the code of player Class.
url: https://github.com/string1995/eee102/blob/master/as2/ex2/player.cpp

ex2/container.cpp
This file contains the code of container Class.
url: https://github.com/string1995/eee102/blob/master/as2/ex2/container.cpp

ex2/archer.cpp
This file contains the code of archer Class.
url: https://github.com/string1995/eee102/blob/master/as2/ex2/archer.cpp

ex2/mage.cpp
This file contains the code of mage Class.
url: https://github.com/string1995/eee102/blob/master/as2/ex2/mage.cpp

ex2/swordsman.cpp
This file contains the code of swordsman Class.
url: https://github.com/string1995/eee102/blob/master/as2/ex2/swordsman.cpp

ex2/player.h
This file contains the head code of player Class.
url: https://github.com/string1995/eee102/blob/master/as2/ex2/player.h

ex2/container.h

This file contains the head code of container Class.

url: https://github.com/string1995/eee102/blob/master/as2/ex2/container.h

ex2/archer.h

This file contains the head code of archer Class.

url: https://github.com/string1995/eee102/blob/master/as2/ex2/archer.h

ex2/mage.h

This file contains the head code of mage Class.

url: https://github.com/string1995/eee102/blob/master/as2/ex2/mage.h

ex2/swordsman.h

This file contains the head code of swordsman Class.

url: https://github.com/string1995/eee102/blob/master/as2/ex2/swordsman.h

6. Test

6.1 iFraction

Input Test

Input when declaration, all input should be normalized and simplified.

Test result:

```
iFraction a;
iFraction b(1,3,-4);
iFraction c = convertF(Fraction(3,-4));
iFraction d(-3.14);
iFraction e = 1.333333;
iFraction f = -b;
```

```
Output in iFraction Form:
a: 0
b: -1(3/4)
c: -3/4
d: -3(7/50)
e: 1(1/3)
f: 1(3/4)
```

Figure 3 – Input Test of declaration

Output Test

Output test using cout directly.

Test result:

```
cout << "Output in iFraction Form: " << endl;
cout << "a: " << a << endl;
cout << "b: " << b << endl;
cout << "c: " << c << endl;
cout << "d: " << d << endl;
cout << "e: " << e << endl;
cout << "f: " << f << endl << endl;
```

```
Output in iFraction Form:
a: 0
b: -1(3/4)
c: -3/4
d: -3(7/50)
e: 1(1/3)
f: 1(3/4)
```

Figure 4 – Output Test of direct cout

Output test of integer.

Test result:

```
cout << "Output integer: " << endl;
cout << "a: " << a.integer() << endl;
cout << "b: " << b.integer() << endl;
cout << "c: " << c.integer() << endl;
cout << "d: " << d.integer() << endl;
cout << "e: " << e.integer() << endl;
cout << "f: " << f.integer() << endl << endl;
```

```
Output integer:
a: 0
b: -1
c: 0
d: -3
e: 1
f: 1
```

Figure 5 – Output Test of integer

Output test of decimal.

Test result:

```
cout << "Output in Decimals Form: " << endl;
cout << "a: " << a.val() << endl;
cout << "b: " << b.val() << endl;
cout << "c: " << c.val() << endl;
cout << "d: " << d.val() << endl;
cout << "e: " << e.val() << endl;
cout << "f: " << f.val() << endl << endl;
```

```
Output in Decimals Form:
a: 0
b: -1.75
c: -0.75
d: -3.14
e: 1.33333
f: 1.75
```

Figure 6 – Output Test of decimals

Output test of numerator.

Test result:

```
cout << "Output Numerator: " << endl;
cout << "a: " << a.itop() << endl;
cout << "b: " << b.itop() << endl;
cout << "c: " << c.itop() << endl;
cout << "d: " << d.itop() << endl;
cout << "e: " << e.itop() << endl;
cout << "f: " << f.itop() << endl << endl;
```

```
Output Numerator:
a: 0
b: 3
c: 3
d: 7
e: 1
f: 3
```

Figure 7 – Output Test of numerator

Output test of denominator.

Test result:

```
cout << "Output Denominator: " << endl;
cout << "a: " << a.bottom() << endl;
cout << "b: " << b.bottom() << endl;
cout << "c: " << c.bottom() << endl;
cout << "d: " << d.bottom() << endl;
cout << "e: " << e.bottom() << endl;
cout << "f: " << f.bottom() << endl << endl;
```

```
Output Denominator:
a: 1
b: 4
c: 4
d: 50
e: 3
f: 4
```

Figure 8 – Output Test of denominator

Operator Test

Let b = -3/4. Test the opposite number -b and reciprocal ~b.

Test result:

```
cout << "-b = " << -b << endl;
cout << "~b = " << ~b << endl << endl;
```

```
-b = 1(3/4)
~b = -4/7
```

Figure 9 – Operator Test of – and ~

Let b = -3/4, e = 4/3 and c = 5. Test operator +, -, *, / and %.

Test result:

```
cout << b << " + " << e << " = " << b+e << endl;
cout << b << " - " << e << " = " << b-e << endl;
cout << b << " * " << e << " = " << b*e << endl;
cout << b << " / " << e << " = " << b/e << endl;
cout << e << " % " << c << " = " << e%c << endl << endl;
```

```
-1(3/4) + 1(1/3) = -5/12
-1(3/4) - 1(1/3) = -3(1/12)
-1(3/4) * 1(1/3) = -2(1/3)
-1(3/4) / 1(1/3) = -1(5/16)
1(1/3) % -3/4 = 7/12
```

Figure 10 – Operator Test of +, -, *, / and %

Let b = -3/4, e = 4/3. Test operator +, -, *, / and % interreact with other types of number.

Test result:

```
cout << b << " + " << 4.44 << " = " << b+4.44 << endl;
cout << 4.44 << " + " << b << " = " << 4.44+b << endl;
cout << b << " - " << 3 << " = " << b-3 << endl;
cout << b << " * " << 2.33333 << " = " << b*2.33333 << endl;
cout << 2.33333 << " * " << b << " = " << 2.33333*b << endl;
cout << b << " / " << 5 << " = " << b/5 << endl;
cout << e << " % " << 1 << " = " << e%1 << endl << endl;
```

```
-1(3/4) + 4.44 = 2(69/100)
4.44 + -1(3/4) = 2(69/100)
-1(3/4) - 3 = -4(3/4)
-1(3/4) * 2.33333 = -4(1/12)
2.33333 * -1(3/4) = -4(1/12)
-1(3/4) / 5 = -7/20
1(1/3) % 1 = 1/3
```

Figure 11 – Operator Test of +, -, *, / and % with other type

Let b = -3/4. Test b++, ++b, b--, --b.

Test result:

```
cout << "b++ = " << b++;
cout << "; b = " << b << endl;
cout << "++b = " << ++b;
cout << "; b = " << b << endl;

cout << "b-- = " << b--;
cout << "; b = " << b << endl;
cout << "--b = " << --b;
cout << "; b = " << b << endl << endl;
```

```
b++ = -1(3/4); b = -3/4
++b = 1/4; b = 1/4
b-- = 1/4; b = -3/4
--b = -1(3/4); b = -1(3/4)
```

Figure 12 – Operator Test of b++, ++b, b--, --b

Let b = -3/4 and c = 5. Test assignment +=, -=, *=, /=.

Test result:

```
cout << "b += c; b = " << (b+=c) << endl;
cout << "b -= c; b = " << (b-=c) << endl;
cout << "b *= c; b = " << (b*=c) << endl;
cout << "b /= c; b = " << (b/=c) << endl << endl;
```

```
b += c; b = -5/2
b -= c; b = -7/4
b *= c; b = 21/16
b /= c; b = -7/4
```

Figure 13 – Assignment Test of +=, -=, *=, /=

Comparison Test

Let b = -3/4, c = 5 and f = 3/4. Test >, <, >=, <=, ==, !=.

Test result:

```
cout << "if b > c ?? " << ((b>c)? "Yes" : "No") << endl;
cout << "if b < c ?? " << ((b<c)? "Yes" : "No") << endl;
cout << "if b >= c ?? " << ((b>=c)? "Yes" : "No") << endl;
cout << "if b <= c ?? " << ((b<=c)? "Yes" : "No") << endl;
cout << "if b == c ?? " << ((b==c)? "Yes" : "No") << endl;
cout << "if b == -f ?? " << ((b==-f)? "Yes" : "No") << endl;
cout << "if b != c ?? " << ((b!=c)? "Yes" : "No") << endl << endl;
```

```
if b > c ?? No
if b < c ?? Yes
if b >= c ?? No
if b <= c ?? Yes
if b == c ?? No
if b == -f ?? Yes
if b != c ?? Yes
```

Figure 14 – Comparison Test of >, <, >=, <=, ==, !=

Divide 0 Test

Declare a new iFraction with denominator equals 0.

Test result:

```
try{
    iFraction g(2,1,0);
}catch(const char* msg){
    cout << msg << endl;
}
```

```
Division by zero condition!
```

Figure 15 – Divide 0 Test

6.2 Game

Swordsman Test

Enter the game and chooses to be a swordsman.

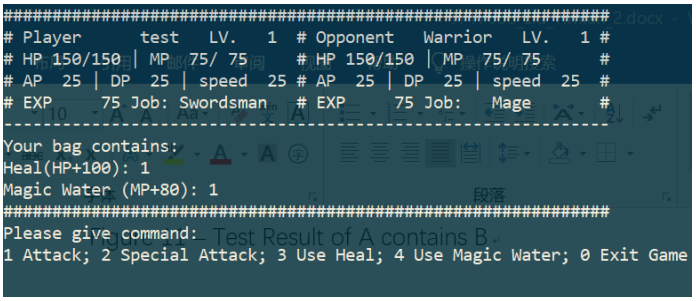


Figure 16 – First Round

As it shown in the Figure, the opponent is randomly selected as a Mage.

Then we choose 1. Attack.

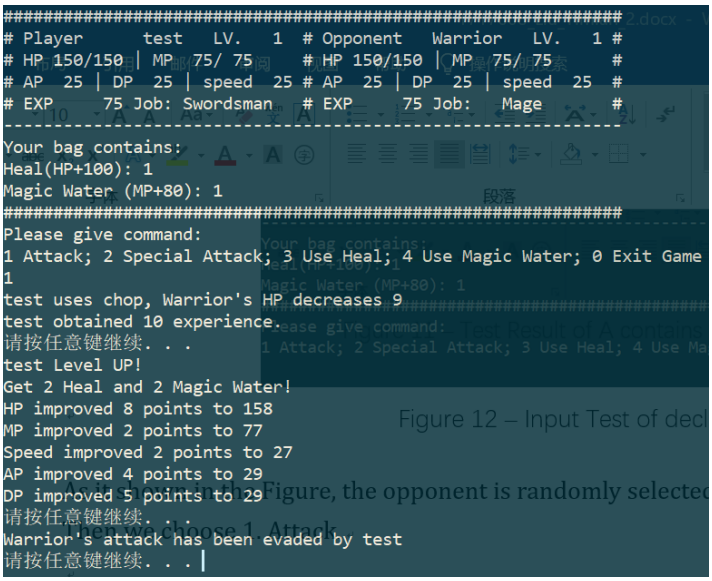


Figure 17 – Attack

As it shown in Figure, the player uses chop and the warrior's HP decreases 9, which the player get 10 experience. Then, the player level up, obtain 2 heal and 2 magic water. And also obtain HP, MP, Speed, AP, and DP. Then the opponent try to attack the player but the player evade it because of the high Speed as well as lucky.

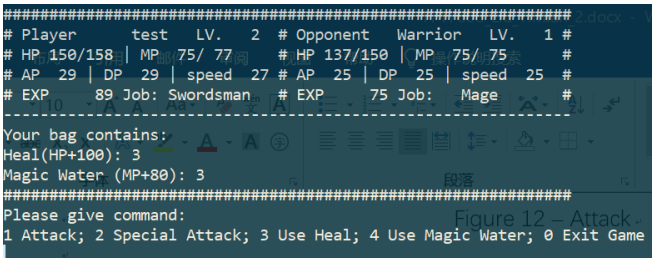


Figure 18 – Next Page

From the figure above, we can see that the level of the player had increased to 2, while obtain 3 Heal and 3 Magic Water, at the same time other characters also somewhat improved.

Then we test 2. Special Attack.

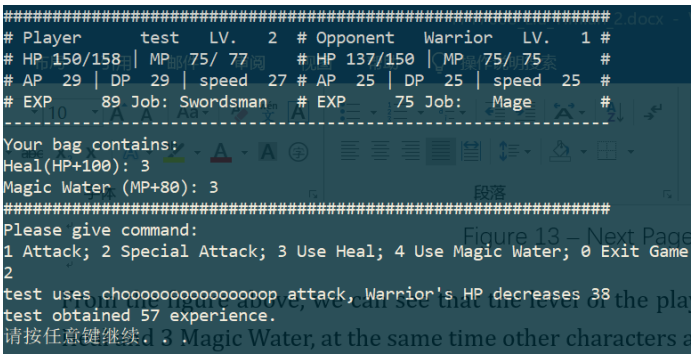
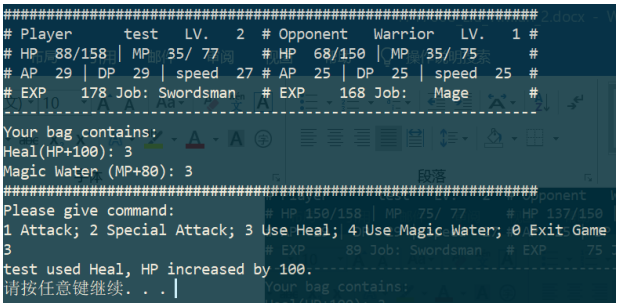


Figure 19 – Special Attack

From the above picture we can find that a special attack named choooooooooop had been released. Meanwhile, the Mp of player decrease while the Hp of opponent sharply decreased.

We the test the heal.



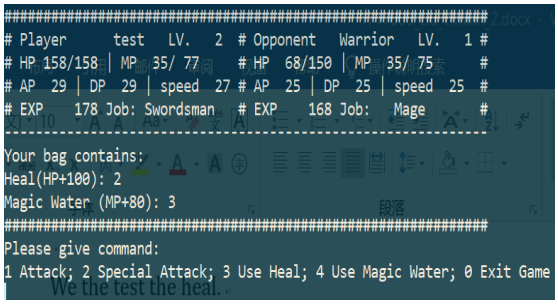


Figure 20 – use Heal

From the last figure, we can find that after using Heal, the Heal decrease 1 and HP increased.

Then we test Magic Waters.

Figure 21 – use Magic Water

As it shown in last figure, after using Magic Water, the magic waters decrease 1, while the Mp increase.

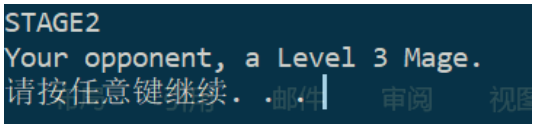


Figure 22 – Upgrade

Last figure shows the situation of upgrading.

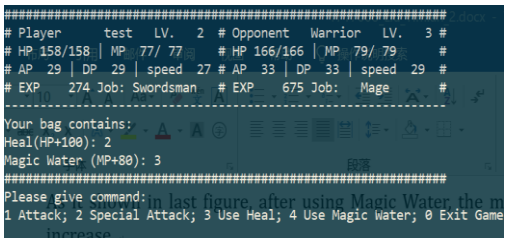


Figure 23 – After Upgrade

Last figure shows the situation after upgrading.

Archer Test

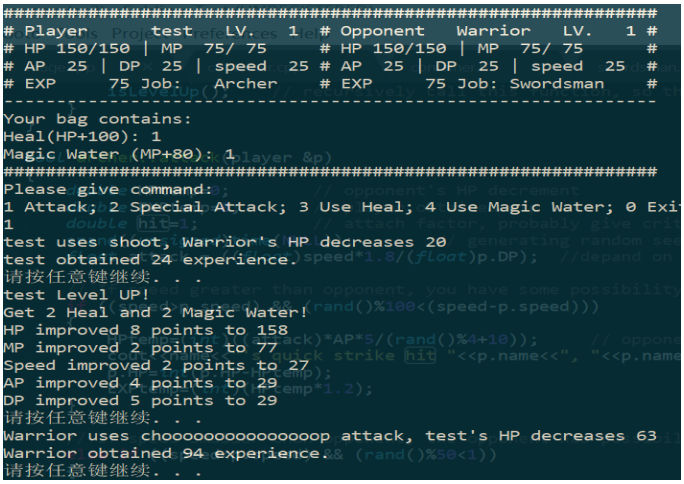


Figure 24 – Archer Test

As it shown in last figure, the attack of archer is shoot, which can cause around 20 HP decrease to a DP 25 opponent

.

Mage Test

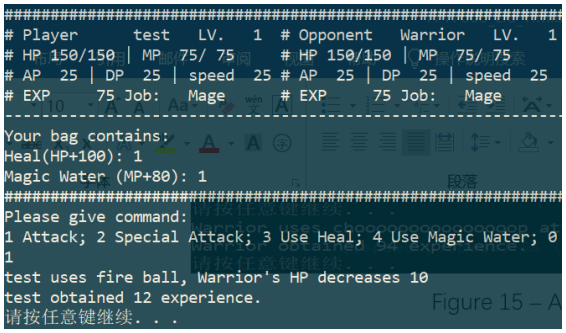


Figure 25 – Mage Test

As it shown in last figure, the attack of mage is fire ball, which can cause around 10 HP decrease to a 25 DP enemy.

7. Conclusion

In this report, two exercises had been detailly conducted, analyzed, designed and tested. From this assignment, we had designed a subclass of Fraction class from last assignment to achieve the functions of mixed fraction. From this exercise, we had obtained the skills of inherit as well as a basic understanding of the relationship of classes and functions. Then in the second exercise, we are required to modify a game code, which gave me an overall scope of how an object-oriented project looked like. And also have a general background about how a game could be generated through C++ project. In a word, from this assignment I have learnt a lot and began to cultivate a sense of cooperation and project management.