# C Project REPORT: MAGIC tower

## Abstract

In the process of completing the project, we use C language as a development tool to build this project and in the middle of this semester we finished the fundamental functions of this project which means this game became playable. In the second part of this semester, we finish our maps design and polish up the game by adding several functions like shop function and damage function.

This report will talk about our whole semester’s work: Magic Tower from six levels: Introduction/Problem statement, Group Division, Analysis, Design, Implementation, Testing/Debugging and Conclusion. By using pictures, flowcharts and charts, we try to make this report more readable and more explicit.

## Introduction/Problem Statement

The Magic Tower game is a generic term for puzzle-type electronic role-playing games, originated in a Japanese PC-9801 MS-dos game "Tower of the Sorcerer"(“魔法の塔” in Japanese).

Magic Tower is really a non-linear puzzle game disguised as an RPG.

It appears to be RPG-style; you have attack strength, defense strength, and a health bar, and so do all the monsters. You can increase your health (without limit) by drinking potions and increase your attack and defense strength by collecting crystals and improved equipment, as well as by donating gold to the altars on various floors. However, since the combat is completely deterministic, it is actually a puzzle.

When you engage a monster, you take turns hitting each other until one falls (the player goes first, except during some of the boss fights). There is no random element; the amount of damage is simply attack strength minus defense strength. It is a normal occurrence for one to have insufficient attack power to hurt the other.

Thus, your strategy is based on choosing the order of your battles so as to progress through the levels, conserve your health (and keys), and maximize your attack and defense strength. To assist in this, you are given the Orb of the Hero, which shows you the capabilities of the monsters on the current floor and the amount of damage that each would inflict if you were to challenge them. Your exact path is then up to you.

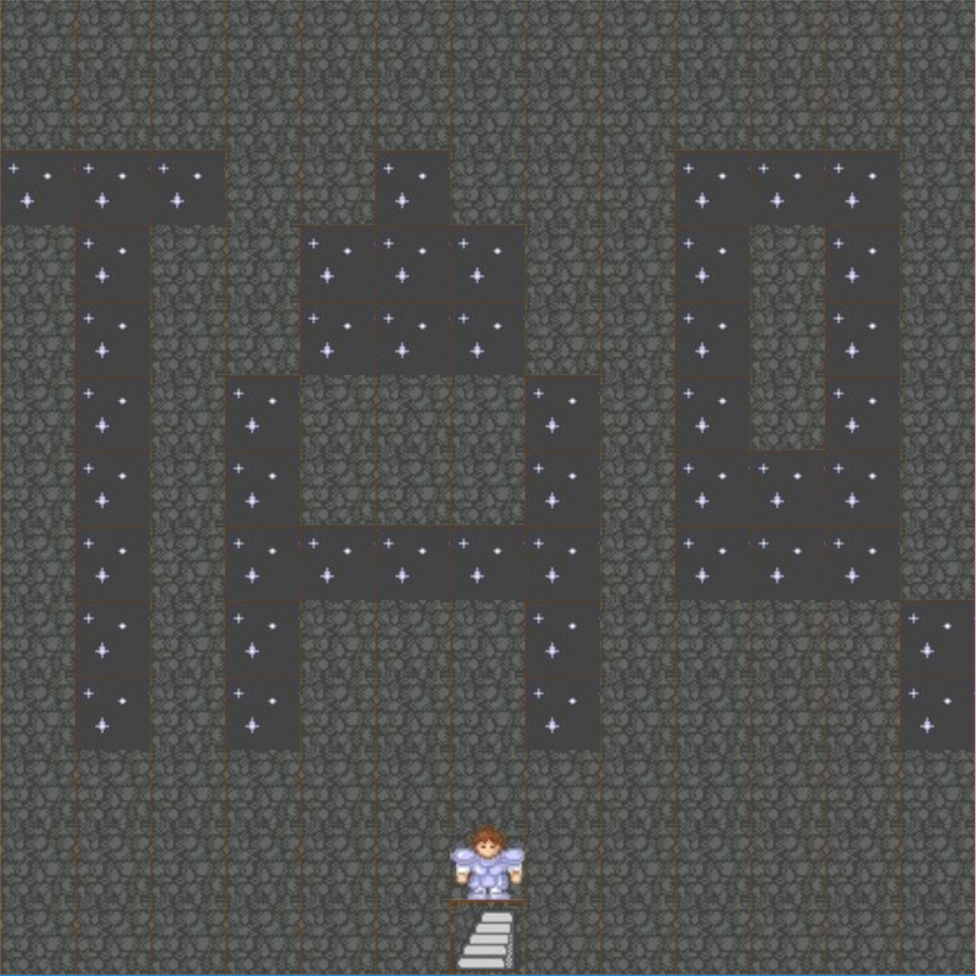
## Analysis

We require:

1. a map designed logically enough that there is a way to get through the game;



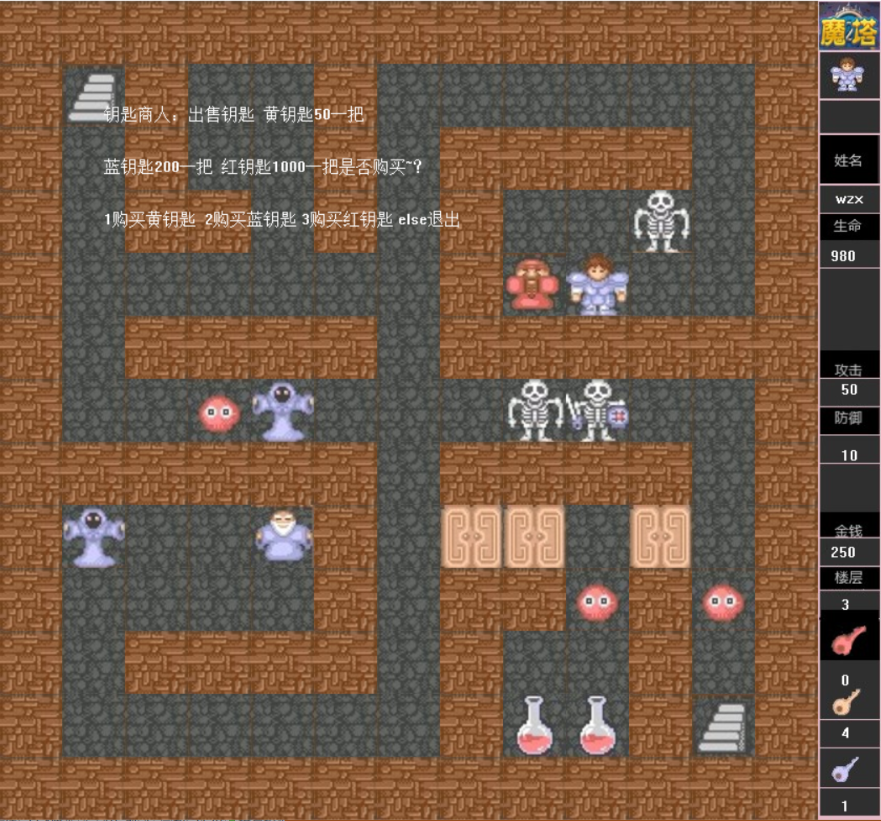
1. to judge the element the prince meets and according to it do a certain thing. For example, go upstairs when meeting an upstairs ladder;



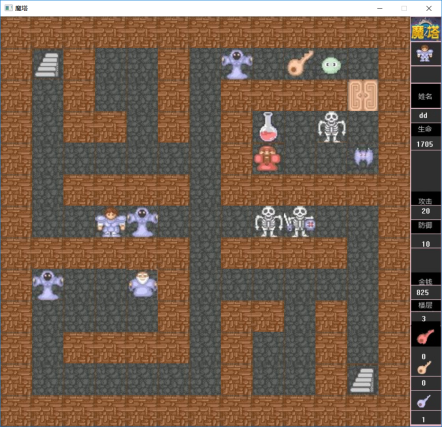
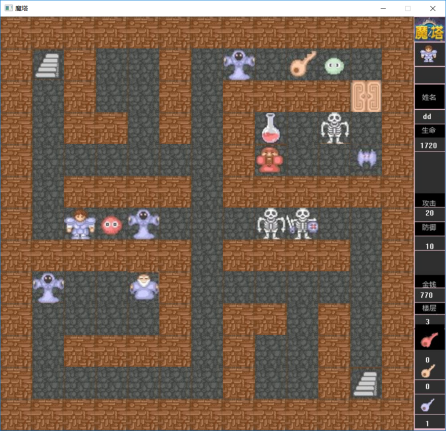
1. to move controlled by the keyboard. That is, go ahead when entering “w”, go back when entering “s”, turn left when entering “a” and turn right when entering “d”;
2. to save the game so that the player can continue to play the unfinished game. That is, when the player chooses to enter the saved game, he can go somewhere like this directly;



1. to show the dialogue when meeting an NPC;



1. proper parameters for playability, such as the attack and defense strength of each monster;
2. to calculate the losing health point(HP) for each monster (according to the property of the player). For example, when the prince meets a monster as following, he will lose some Hp.



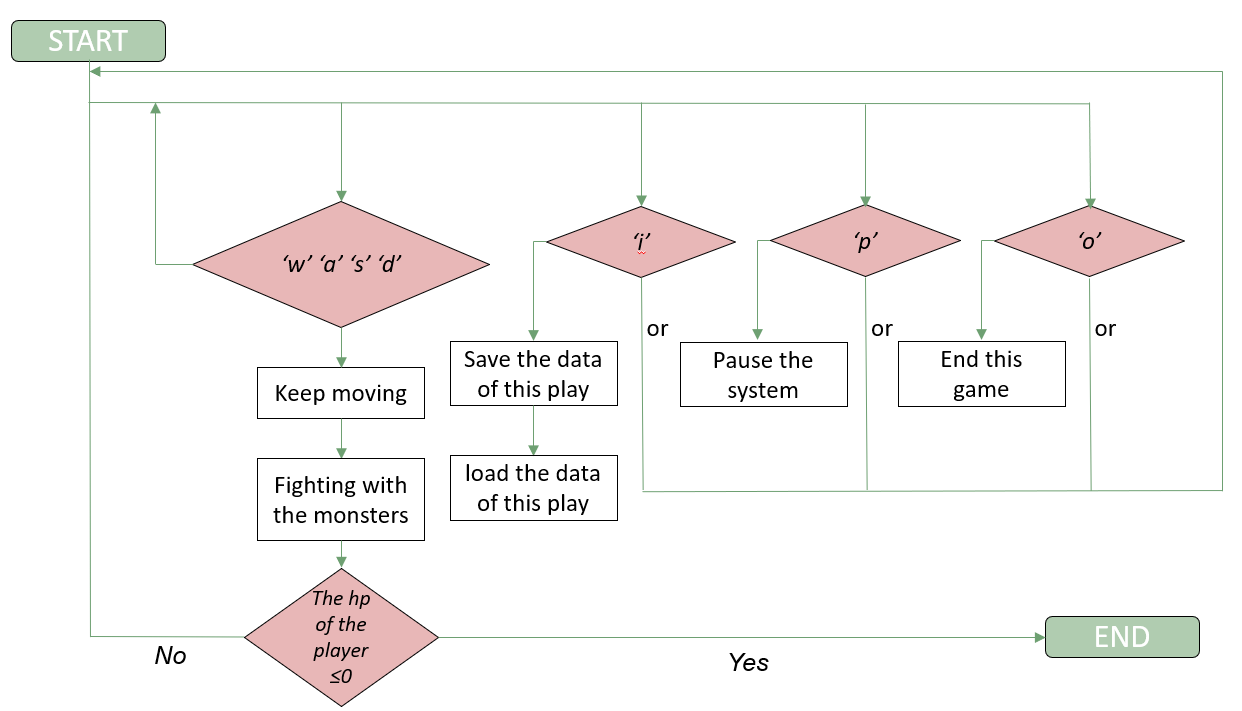
1. to give some property, such as a key or attack strength, when the player buys it in a shop;



## Design

In order to realize the purpose to make the game graphical and enable the judgement and response about the key that is entered by our players, we put forward the following project design to finish those main parts using some flowcharts to illustrate our design.

①About the key judging:

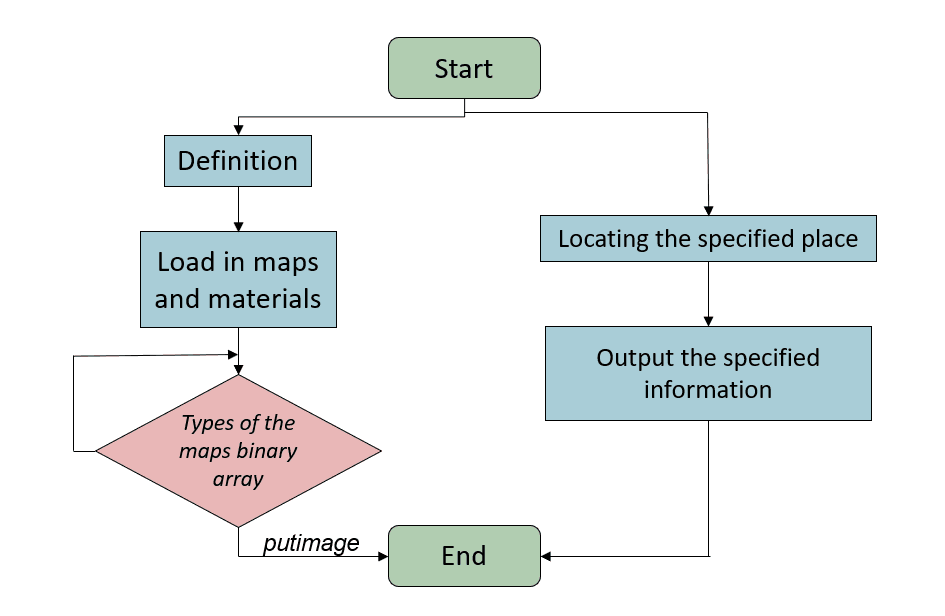


At first, we talk about how many types of keys can we offer to our users to play this game. So we design four kinds of keys for computer to judge. We then offer ‘ w a s d ’ for the player to move the prince, ‘ i ’ for the player to save the process of this time’s game, ‘ p ’ to pause the interface so that player can leave for a while to do other things and the last one ‘ o ’ to exit this game so that our player can restart the game and make new strategy to finish the adventure.

Next, since the ‘ i ’, ‘ p ’ and ‘ o ’ which only need an easy response from the system is not difficult to realize, we think that the most important part is about the player moving, because this order relate the place of the prince and the kind of the next room for the prince. We need to design and make a function to judge the next kind of the room and let the prince gain the good effect or bad effect based on the type of the next room. After all this series’ judgement is finished or the player enter the other illegal keys, the circle will come back to the first part.

The last but not the least, after the prince gain the effect like losing hp or gaining some buff, the system should always judge whether the hp of our prince is lower than zero. If the hp is lower than zero, this game will finish and the system will come to the next interface to let the player exit.

②About the graphical interface:



In order to come true our project, we ever consider lots of ways to present the interfaces of our game. However, because magic tower is a player-play game, what we need to come true is to let the soldier move on the interface. Therefore, we need to realize the graphical interfaces. We understand that we may lack the ability to design basic appearance of so many kinds of rooms, so that we choose to load the materials of rooms from the internet. What is more, we are assisted by the library of graphic.h. And then we design 13 maps for this adventure. What’s more, we use binary array to design the maps to load the material.

We divide this objects into two parts ‘ pictures ’ and ‘ texts ’ to appear in the interface.

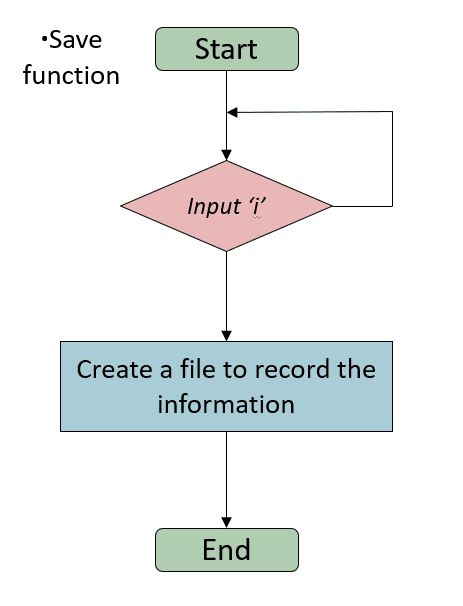
For ‘ pictures ’ part, at first we define many kinds of the room represented by numbers. Then we use binary array to load and show our maps. Based on the number of the binary array, we use different pictures to put the image.

For ‘ texts ’ part, we use ‘gotoxy’ and ‘ outtext ’ to output the texts in the right coordinate.

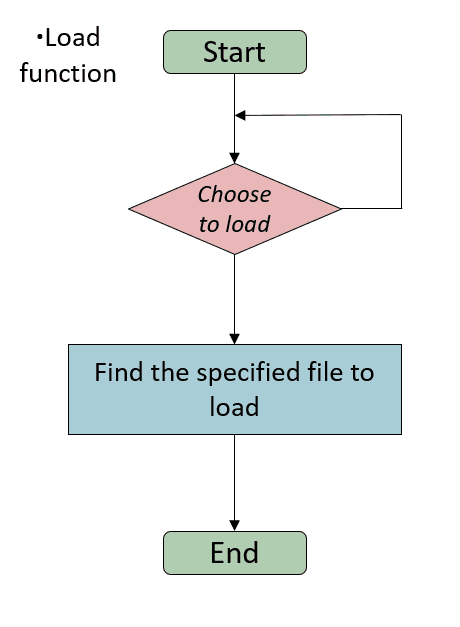
③About the saving and loading part:

In the process of the game, the player can enter the key ‘ i ’ to save the process, so that when he begin the game next time, he can load the file and come back to the original state to continue the game.

For the saving part, we can create a file called ‘ mtw ’ at first and then save all the data in this file.



For the loading part, we find a file named ‘ mtw ’ and load all the data from the file to give every variable a value so that the player can come back to the last game.



④About the damage function:

In the moving part there exists a possibility that the prince should fight with so many monster in the way to save the princess. So that the damage function is really important.



In this part, we should let the damage function come back with a value to decline the number of the prince which is based on the hp of the monster, attack of the monster, defence of the monster and the same attribute value of our prince.

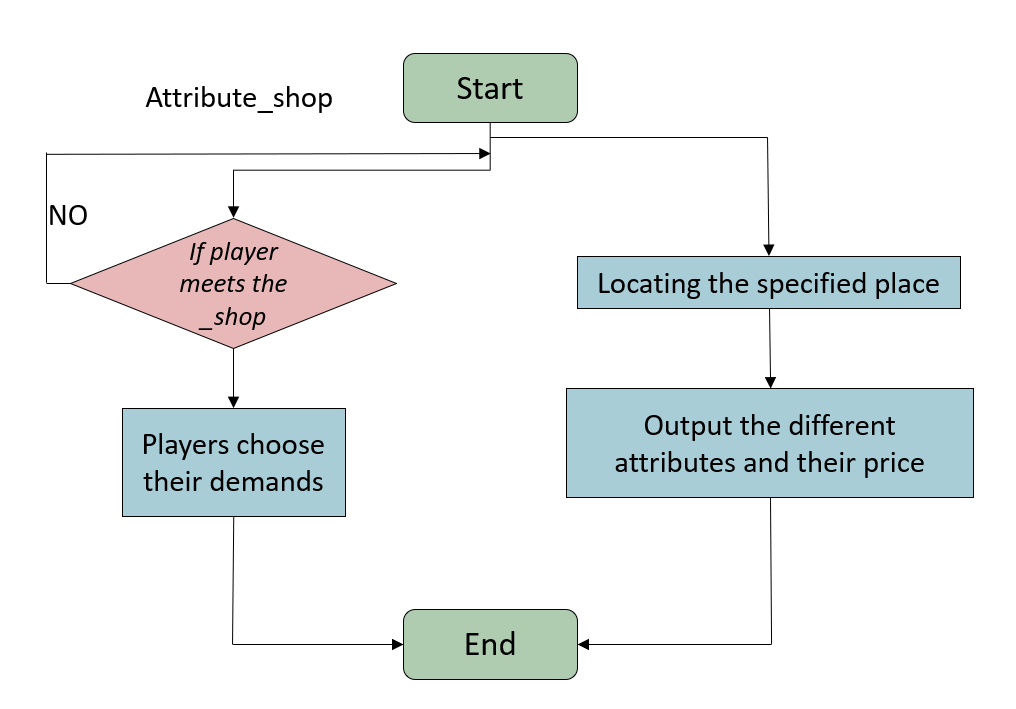
And if the attack of our prince is lower than the the defence of the monster which means the prince can never make any damage to the monster, we will let the prince to lose all the hp and exit the adventure interface.

⑤About the key shop and the attribute shop:



To design the key shop, we need to divide this function to two parts, one thing is the output parts, we should show the information of the price of the key in the game interface when the prince meet the key merchant.

For another thing, we should gain the kind of the key that is entered so that we can come to the next judgement whether the money of the prince is enough to buy this kind of the keys like yellow keys ‘50’ blue keys ‘200’ and red keys ‘1000’.



To design the attribute shop, we need to also divide this function to two parts just like the key show, one thing is the output parts, we show the price of the exercises which give different buff to increase the attribute value like attack and defence of the prince in the interface of the game when the prince meet the attribute merchant during the process of the game.

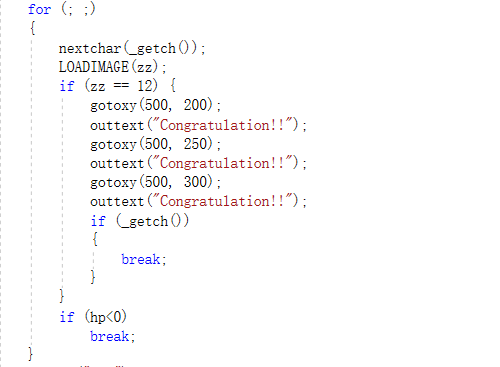
For another thing, we also should gain the kind of the key that is entered and judge the money of the players to gain the attribute.

## Implementation

According to what we have said in the part of design, in this part we will explain our codes and documentation of the problem solution.

①About the key judging:

First we talk about the sentences about key judging in the main function.



We use a for sentence to let the game always continue based on the judging sentence in the flowchart above. In the for sentence, we use a ‘nextchar’ function to gain the key that is entered by the player. Then according to the effect that is gained depending on the kind of the next room. We need to renew the map and the interface. So we write a LOADIMAGE function in the next line and the variable ‘ zz ’ is the number of the floor that the prince is moving on. The next part if sentence is judging whether the player finish the adventure and come to the last floor to save the prince. If it is, then we will write three words ‘Congratulation’ to congratulate the winner. The last part is about the judgement of whether the ‘ hp ’ is lower than zero just like the flowcharts showed above. When ‘ hp ’ is lower than zero, this game will be over. So we write a ‘ break ’ in the last line of this part.

Then we talk about the parts of the judgement of the next rooms. There are two functions linked to this part: ‘nextchar’ and ‘nextrooms’.

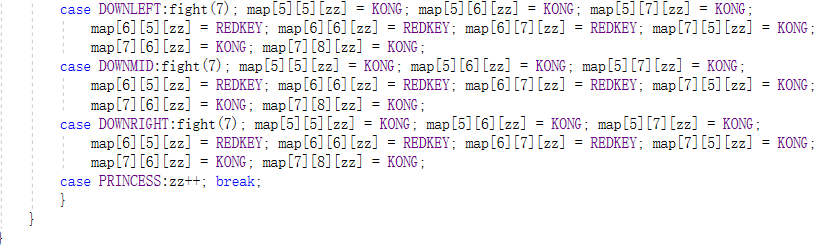
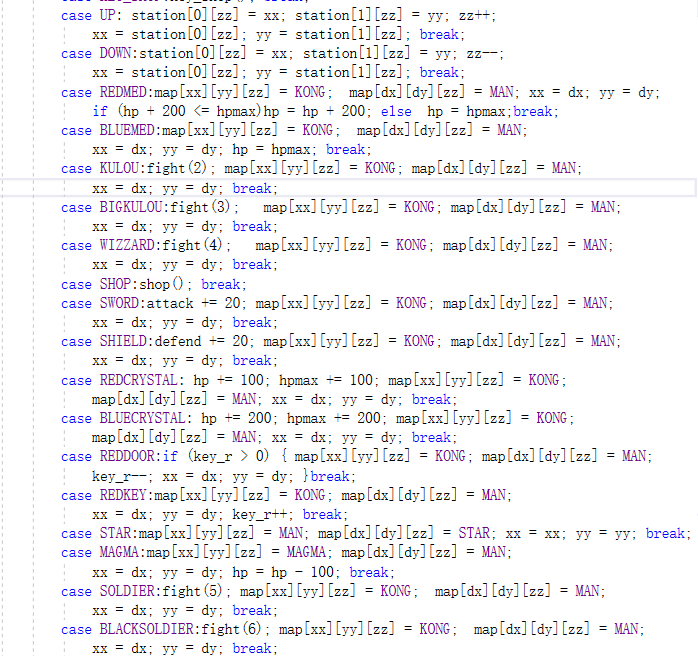


The ‘ nextchar ’ function is designed to judge the different kinds of the key that is entered by the player. If it is ‘ w s a d ’ which means the player wants to move the prince, we will come to the function ‘ nextroom ’ which is designed to gain the effect of the next room.

If the key is entered ‘ p ’, we write a ‘system(“ pause”)’ to pause the interface and the process of the game. If it is ‘ i ’, then we will link the ‘ save ’ function which will be explained below to save all the data so that the player can continue the game next time. If the char is ‘ o ’, this int function will return ‘ 0 ’, so that the player can exit the game.

After all the sentences has been executed, the int function will return a number ‘ 1 ’ .Then the execution will go on in the for sentence in the main function that is showed above.





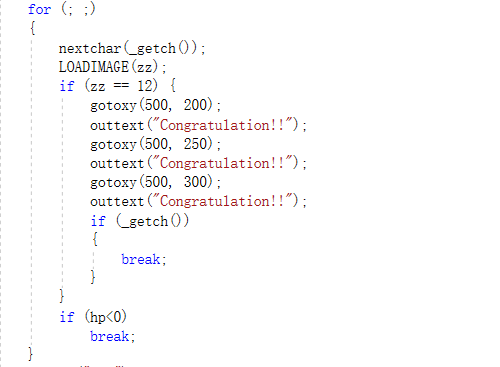
The most important core of this object function ‘ nextroom ’ which judges the type of the next room and make change to the place of the prince, gain the damage of the monster, open the door based on the keys that the prince has or many other possibilities.

At first we declare a two variable ‘ dx ’ and ‘ dy ’ to help us gain the coordinate of the next room that the player wants to move to and gain the number of the next room that is recorded in the binary arrays of maps to gain the effects based on different cases.

The beginning part of the switch sentence is to map the coordinate of the next room using the variable ‘ dx ’ and ‘ dy ’ that is declared above.

The next part is to gain the effect based on the kind of the next room. Because there are so many possibilities and many cases has the similar codes, we just use several representative cases to introduces the core idea of this function.

The first one is case YELLOWKEY. YELLOWKEIf the kind of the next room is YELLOWKEY, then the value of the map[dx][dy][zz] which represents the place of the nextroom will be changed to ‘ 1 ’ which is the number of the prince and the value of the map[xx][yy][zz] which represent the place of the prince will be changed to ‘ 0 ’ which is the number of the floor .(We define 0 as KONG in the header file.)And the value of the whole valuable ‘xx’ and ‘yy’ which is the coordinate of the prince will be changed to the coordinate of the next room. And the effect of this kind of the room is to let the number of the yellow key gain one more. And when the process come out into the main function as below which will run the next sentence ‘ LOADIMAGE ’ . The interface will be renewed based on the new binary arrays that has be changed. So that the place of the next room that was a yellow key will be replaced by the prince(MAN) and the place of the room that was prince will be replaced by the floor(KONG).There are other similar case like case MAGMA to move by, case BAT to fight with monster and case UP and DOWN to change the floor of the prince so that the prince can come to other floor.

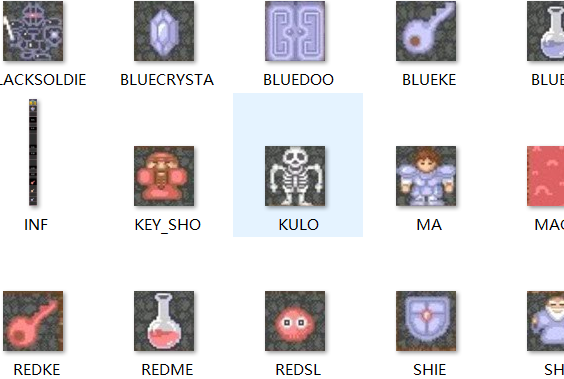


The second one is case WALL.WALBecause we should set the obstacle barrier to limit the way of the player,we have to set a possibility that we prince can never get to the next place.So we didn’t change the value of the map[dx][dy][zz] and map[xx][yy][zz] so when we come back to the sentence ‘ LOADIMAGE ’.The interface will not change.Case STAR and PRINCESS are similar in some ways.

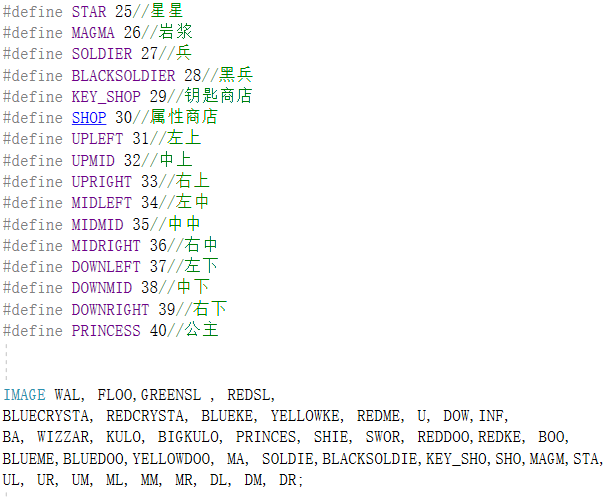
The last one is case KEY\_SHOP.KEY_SHO This case represent so many other cases that use other functions like key\_shop, shop, fight or other functions.When the next room is the key merchant,the execution will run the function key\_shop,so that the player can enter some chars to gain the different keys.

②About the graphical interface:

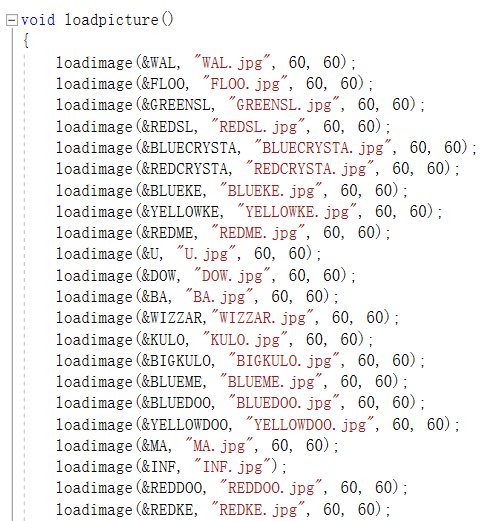
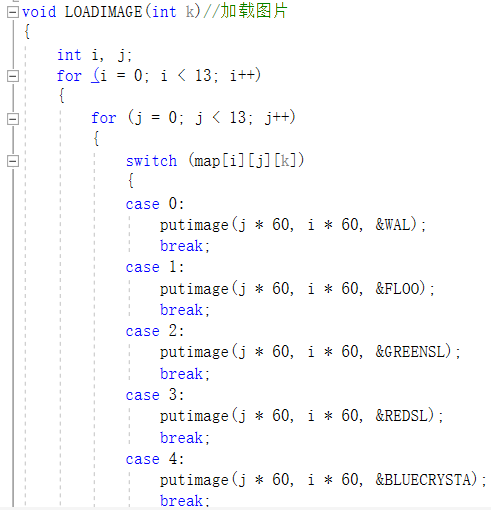
As you can see in the design of the graphical interfaces, we searched the data and material what we need from the internet. You can understand it from the following picture:



Then, we gave the specific definitions of the material of the project:

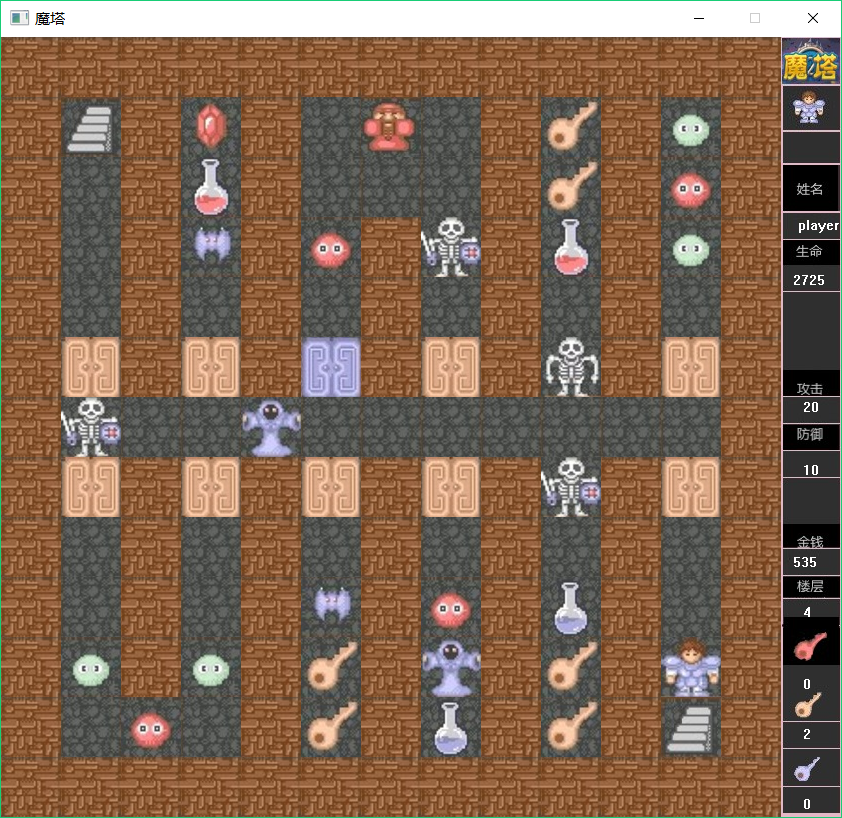
 

Afterwards, we need to put this material and data into the project. Therefore, we mainly use loadimage and putimage to realize this willingness:

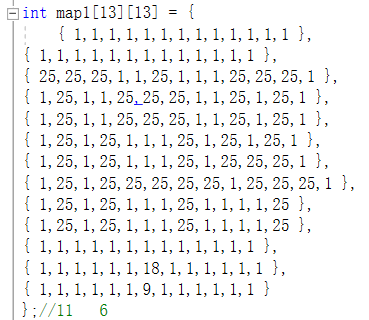
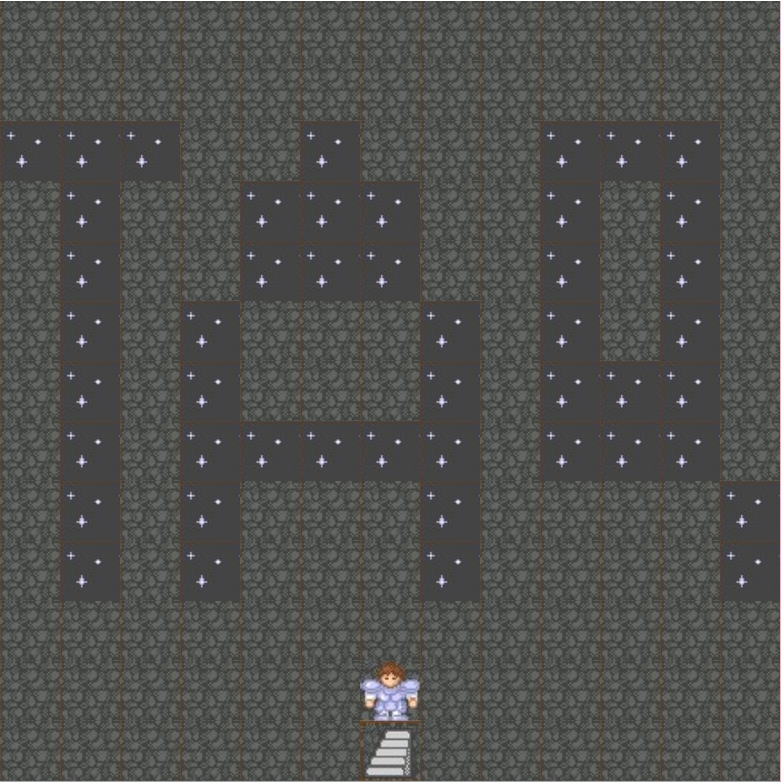
 

(the above pictures are just parts of our codes of these functions)

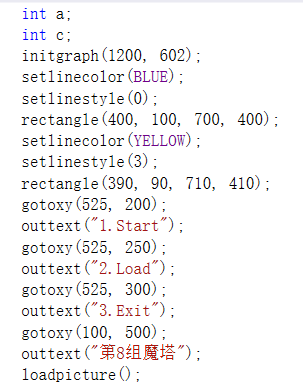
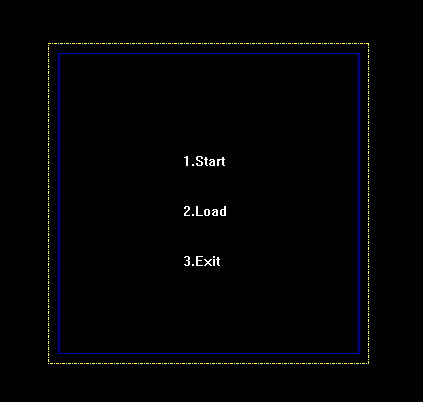
Then, let me show you a finished map’s picture:

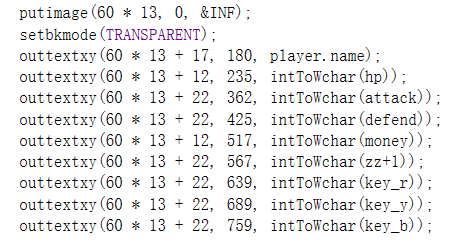
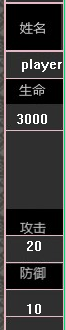


Next, from the design part, we can know the maps are consisted of the binary arrays. We take the first map as an example and you can understand it:

Finally, in the graphical interfaces’ part, we want to output the specified information like pictures and texts, we use outtextxy function to realize it. You will clearly know it from the next pictures:

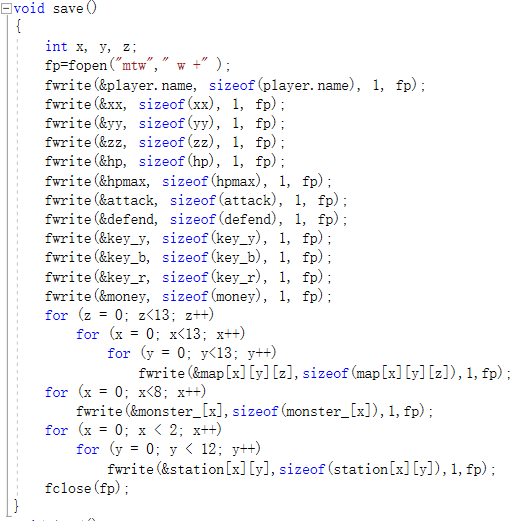
 

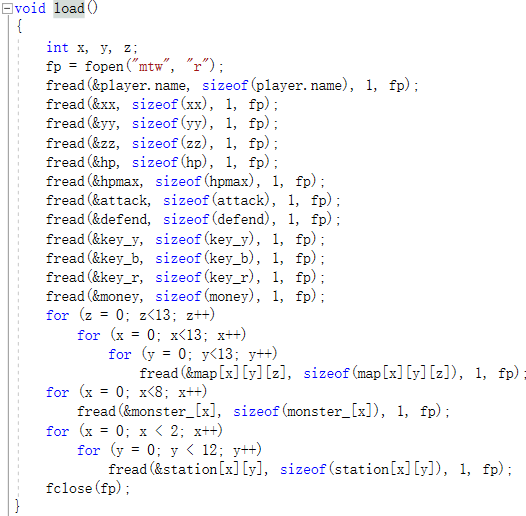
③About load and save:

This two functions is easy to realize if we have a good knowledge of file system and so many other functions like fopen, fread and so on.

The codes is put below:



In this function we declared three variables to help the for sentence and use fwrite to record every data from the valuables like ‘ xx ’, ‘ hp ’ and so on. The last sentence is ‘ fclose ’ to close the file.

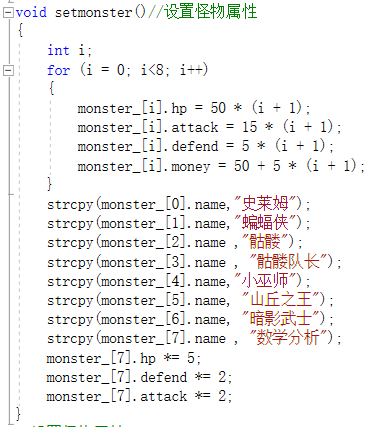


In the begin of the function, we also declare three variables to serve the for sentence

And then we just use the fread to read the data from the file to gain the value of variable. The last sentence is always the ‘ fclose(fp) ’.

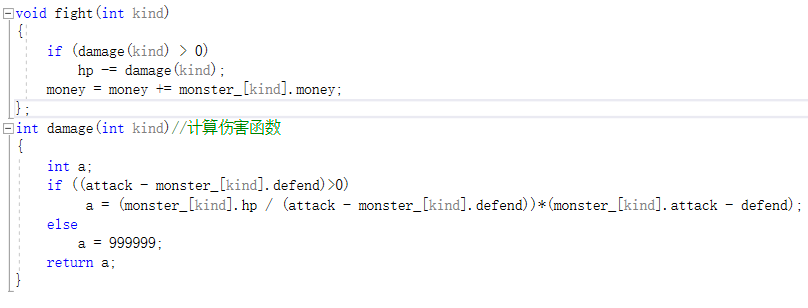
④About the damage function:

In this part, we talk about three functions related: fight, damage and setmonster.



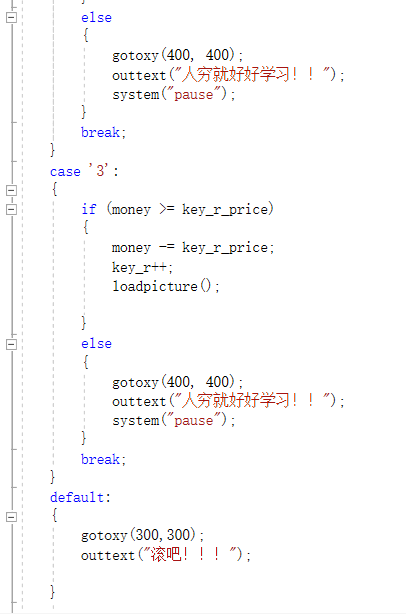
The first one setmonster is to give values to the attribute of so many different monster. This function is used in the begin part of the main function

And when the prince meets a monster the execution will use the function fight and the damage to return a value to let the hp of the prince decline.

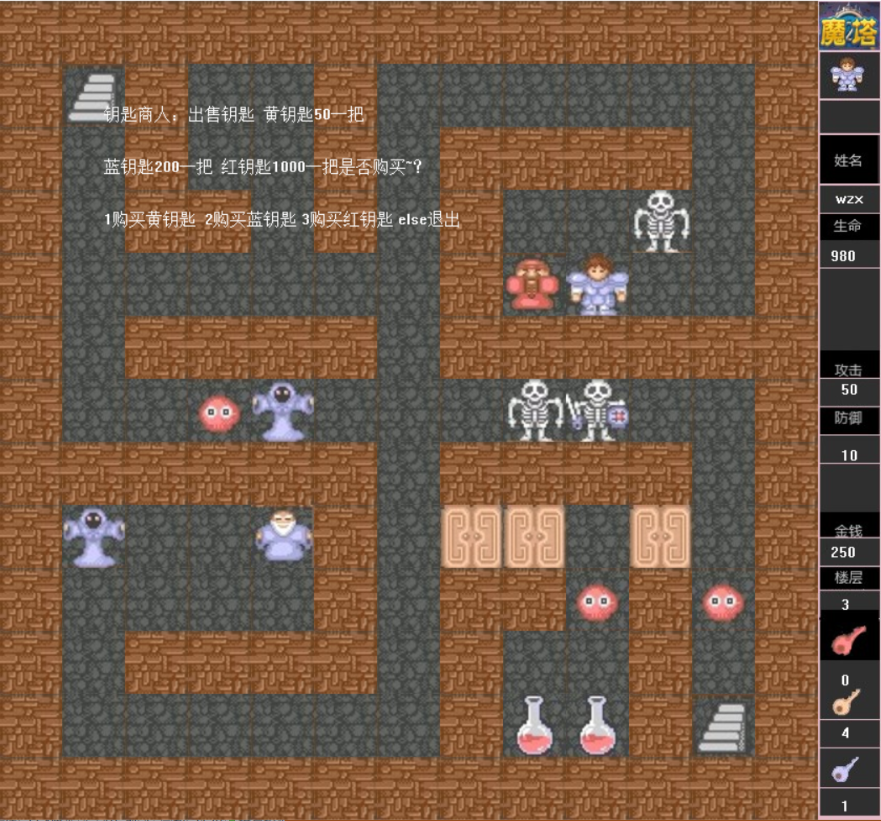


Fight function will get a number which represent the kind of the monster and according to the kind of the monster, we use a damage function to gain the damage. Just like the flowchart that shows above, we use a if sentence to judge whether the attack of the prince is higher than the monster’s defence. If it is, we use a mathematical formula to get the damage. This formula first get how many times will the prince attack the monster to beat the monster and then multiply the damage that the monster will take to the prince. Then we can get the value. If the hp is not higher than the defend of the monster, it means that the prince can never beat the monster so that we return a number 999999 to end this game.

⑤About the key shop and the attribute shop:



We just talk about the key shop and the attribute shop is similar in some ways. Just as what in the design part we have said, we divide this object to two parts: the output interface and the judgement. The first one is easy to present, we just use gotoxy and outtext functions to finish this part. And the interface is as follows:



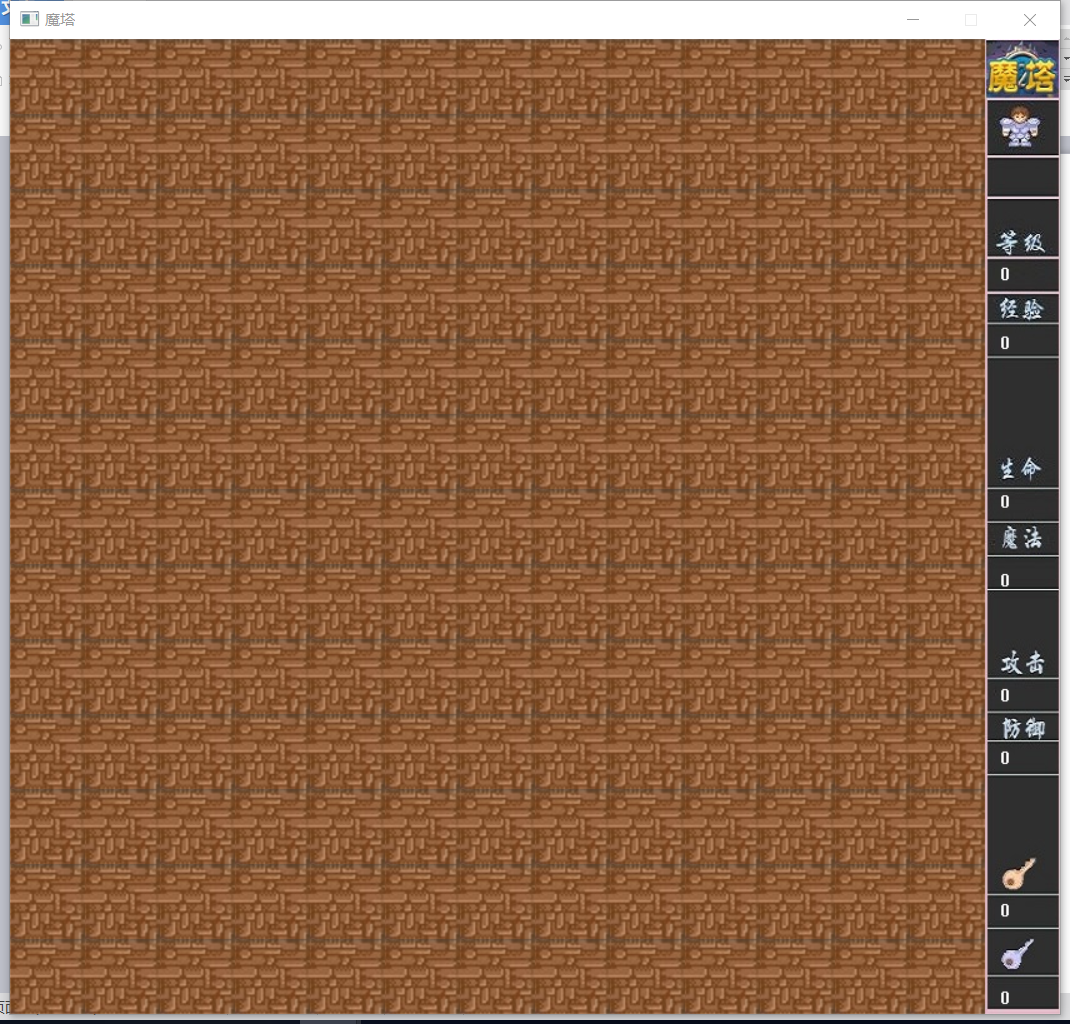
The other part judgement has three possibilities: ‘1’ for the yellow keys, ‘2’ for the blue keys and ‘3’ for the blue keys. All the things we just need are to use a if sentence to judge the char that was entered and judge whether the prince has enough money.

## Testing and Debugging

|  |  |  |
| --- | --- | --- |
| Input | Expected Result | Real Result |
| 1 on the numeric keyboard | Enter a new game | Maps disappear |
| 1 on the main keyboard | Enter a new game | Enter a new game |
| ‘w’ | No wall: the prince moves up for one time  Else: remain there | No wall: the prince moves up for one time  Else: remain there |
| ‘a’ | No wall: the prince moves to the left for one time  Else: remain there | No wall: the prince moves to the left for one time  Else: remain there |
| ‘s’ | No wall: the prince moves down for one time  Else: remain there | No wall: the prince moves down for one time  Else: remain there |
| ‘d’ | No wall: the prince moves to the right for one time  Else: remain there | No wall: the prince moves to the right for one time  Else: remain there |
| Move to the monster | 1. Monster will be killed and disappear. 2. Bonus money will be added on the prince stats. | 1. Monster was killed and disappeared. 2. Bonus money didn’t add. |
| Back to the previous floor | Only one prince | Some princes |
| 2 one the main keyboard | Load the file | No reflection |

**Case 1**

There are numeric keyboard and main keyboard that players can use to type “1”. If players type “1” on the numeric keyboard, the map will not be displayed and players couldn’t play. (Pic\_1) But when players type “1” on the main keyboard, players can start a new game. We found that players can only use the keys on the main keyboard or it will not be read.



(Pic\_1)

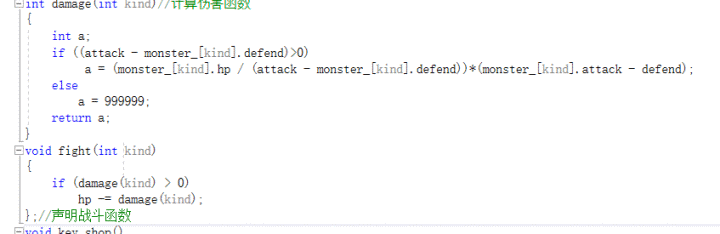
**Case 2**

Players use ‘w’ ‘a’ ‘d’ ‘s’ to control the prince to move randomly. If there is no wall in front of the prince and players type ‘a’ to control the prince, the prince moves up for one time. When move prince to other directions, it’s in the same way.

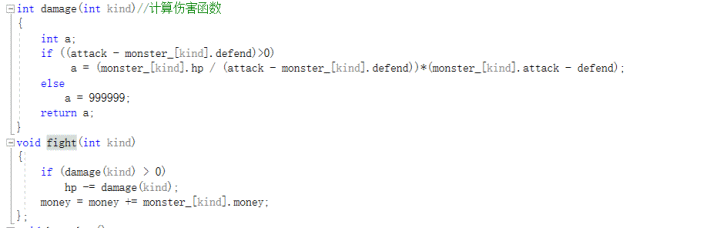
**Case 3**

Under the case that prince’s health point (hp) is enough, after players move prince to the monster, the monster will be killed and disappeared and the bonus money will be added on the prince stats. But when we test we found that after the monster has being killed, the bonus money didn’t add.

The reason is that we didn’t write ‘money increasing’ in the fight function we first define. (Pic\_1) And here are two compared pictures (Pic\_2 and Pic\_3):



(Pic\_2: the fight function without ‘money increasing’)



(Pic\_3: the statement in the red circle is ‘money increasing’.)

**Case 4**

When players enter the next floor and want to back to the previous floor, there may be a problem. If this time the prince’s place is different from that players entered this floor, there will be more than one prince after players go back to the previous floor. (Pic\_4)



(Pic\_4)

The reason we found is that the function after players back to the previous floor is not correct enough. Here are the flowing two pictures, one is wrong (Pic\_5) and the other is after corrected (Pic\_6).



(Pic\_5)

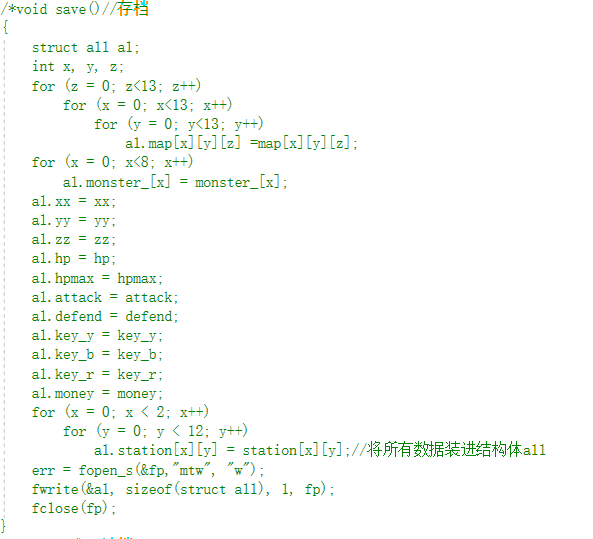
At the first, we only write one possible coordinate of the prince after he goes back to the previous floor and ignore the other possibilities. After correcting, we let stair’s coordinate change into prince’s. And this time, there are no problems.



(Pic\_6)

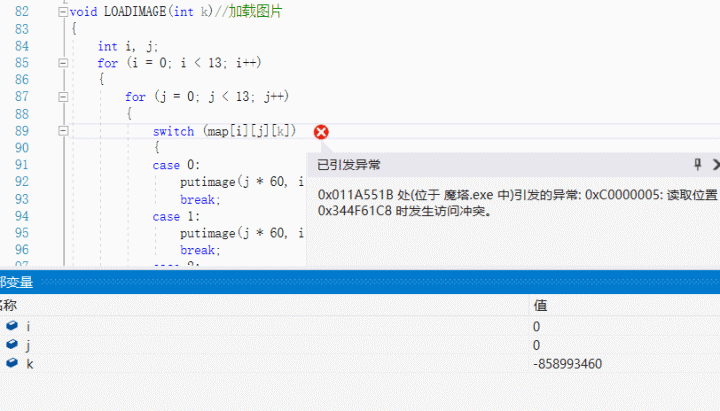
**Case 5**

After we type “2” on the main keyboard, there is no reflection and the file didn’t be read as we expected. The reason is that at first, we want to use structure to record all the information and data as a whole thing and while reading file, those information and data will be read together. (Pic\_7)



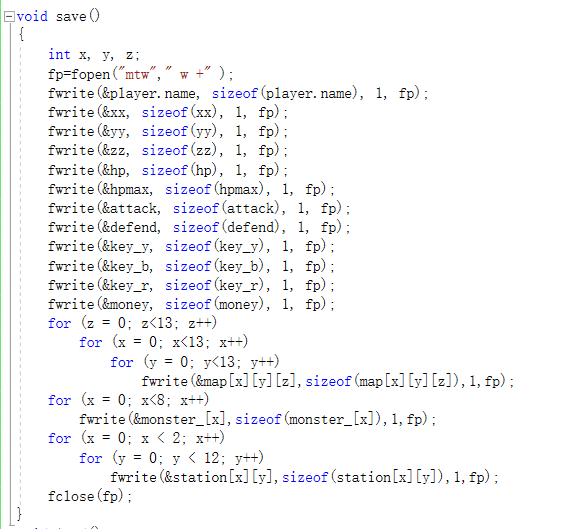
(Pic\_7)

However, when we debug, there comes a problem as the flowing picture shows(Pic\_8):



(Pic\_8)

The reason is that the data we recorded in the structure dislocated and can’t be read in the file we want to load. After we found the reason, we write previous information to the different structures (Pic\_9) one by one and that works.



(Pic\_9)

## Result & Conclusion

After the efforts during the whole semester, our group members have realized the main purpose to build a playable and enjoyable game together. We finished the coding parts, the designing of the maps and the testing and debugging part. In the testing and debugging part we found several problems of our design and document. We can see that the little error of the ‘ seemingly-right ’ idea can never be found only when we try to examine it during the part of execution. Distributing properly and creating a culture of team spirit is the way to make the project finished more effectively.

In fact, there existed several arguments about how to realize those objects put forward above. But if we focus our own job and try to communicate with each other, we can realize the common purpose to manage it. The process of building something together may be a good memory of us forever.