

EEE102 C++ Programming and Software Engineering II

Assessment 3

Report

Prepared by: Xuhui Gong

ID Number: 1718128

April 24, 2019



Xi'an Jiaotong-Liverpool University

西交利物浦大學

Abstract:

This report aims to explain the software development process (SDP) in the two main programs which are vectors comparison and fraction manipulation program. For the first step of SDP is problems specification which focus on the basic problems. Then, for second step is analysis, it focus on the how to write a function to solve the iFraction. Thirdly, it is the most difficult part which is algorithms design which needs to develop a list of steps to solve the problem. Using the algorithms to achieve on code is the fourth step. Finally, testing and verification is also important part, bugs will be found in this part, then the program writer will be fix the bugs.

Contents:

1. Introduction.....	4
2. Problems Specification	4
3. Analysis the Problems.....	6
4. Algorithms Design.....	9
5. Implementation.....	26
6. Testing and Verification.....	27
7. Conclusion.....	37

1. Introduction

The exercise aims to build a whole working games. By using the SOP, this report will provides a complete step which are problem specification, analysis problems, algorithm implementation, code and bug analysis to solve the problem.

2. Problem Specification

This program requires us to implement a game called Monopoly. In this game, there are two players, one is a player itself, the other is a computer-controlled player. The game first requires players to input player information, and can restore information by reading and writing files, which means that the game needs to have the function of reading and writing files. Subsequently, the player enters the game. In the game, the player can choose the first hand and the second hand of the game. After the determination, the player can make the appropriate choice. Players use computer generated random number seeds to simulate dice roll, which is generated

in a range of one to six, and then players play in the generated game board. After the dice is thrown, the player's position is properly displayed, and then the player can make appropriate choices according to his position. For example, when a player's position in the dice roll is not occupied by other players, the player can purchase or upgrade, of course, it is feasible to abandon the purchase. In the game, if the player wants to quit the game, the game data needs to be saved immediately, which is part of data persistence. In the game, according to the player's different position, may be fined, the amount of fines depends on the specific location of the opposite player occupied and the specific amount of fines located. There are other games that require players to be suspended once when they enter prison. In fact, this is equivalent to one more round for other players. The end of the game is when the player's money is less than zero. Initially set the player's initial money to 2000.

3. Analysis the Problems

The workload of this project is actually not very large, but the problem is to implement a console application in a language that is not suitable for writing games. Now we will analyze the problems encountered before, and show the results on the implementation of the algorithm. In fact, my first consideration is how to display a more interesting game interface in the console. I don't think it's a perfect way to draw a game map by printing. So after consulting Microsoft's official file library, I learned that Microsoft has an old drawing method in the console—HDC, but it's not very easy to use in CMD command characters. But as far as I know, it's better to use in shells, because shells support more methods. In the implementation of the algorithm, I will elaborate on the implementation of my drawing method. Secondly, in the player class, you should declare a parent class as in the previous job, and then inherit the parent class by adding two classes. Subsequently, the method of computer should also be considered. In fact, in the process of testing, I set the computer should be compulsory purchase until the computer consumed all the money. In the player setting, I started to use two computers to test the game between two computers, so that one can find the loopholes in the program, the other is through different computers. Method We can analyze which game mode has more advantages in this game. In fact, in the 1970s,

a programmer wrote a chess game. It made two computers play each other and then analyzed and summarized the data. This is also the embryonic form of AI. In addition, document reading and writing is a challenging process, because we don't have a long time to summarize knowledge and apply knowledge because of the course schedule. Finally, the analysis of input and output should be summarized: in terms of input, the program should carry out personal information such as the player's name, as well as the choice of players in the game, which means that the player's location and the property purchased by the player need to be recorded. In terms of output, personal information such as player's name should be properly displayed in the game, including player's location and purchase attributes. In addition, the most important element of the game—money also needs to be recorded and output. In the field of document reading and writing, it is also necessary to input and output these information. Among them, a more difficult problem is how to make the program read and write large amounts of data smoothly with only one file.

4. Algorithm Design

4.1.1 Relationship of class

This program uses four classes, the most important of which is the Player class.

In fact, if there are no other classes, the game can also run normally, except that all the words displayed on the command board will be dull. In addition, Map class is used to draw on the console through HDC in Microsoft's own library. Screen class is used to analyze the screen size to adapt the game entirely, and to adapt the display of different resolutions. In addition, there is a file reading and writing class called File_tool. The purpose of this class is to facilitate file reading and writing, and to make the file of the whole game more orderly. The specific functions and methods of these classes are described in detail below, the Figure 1 has shown the UML diagram of this project.

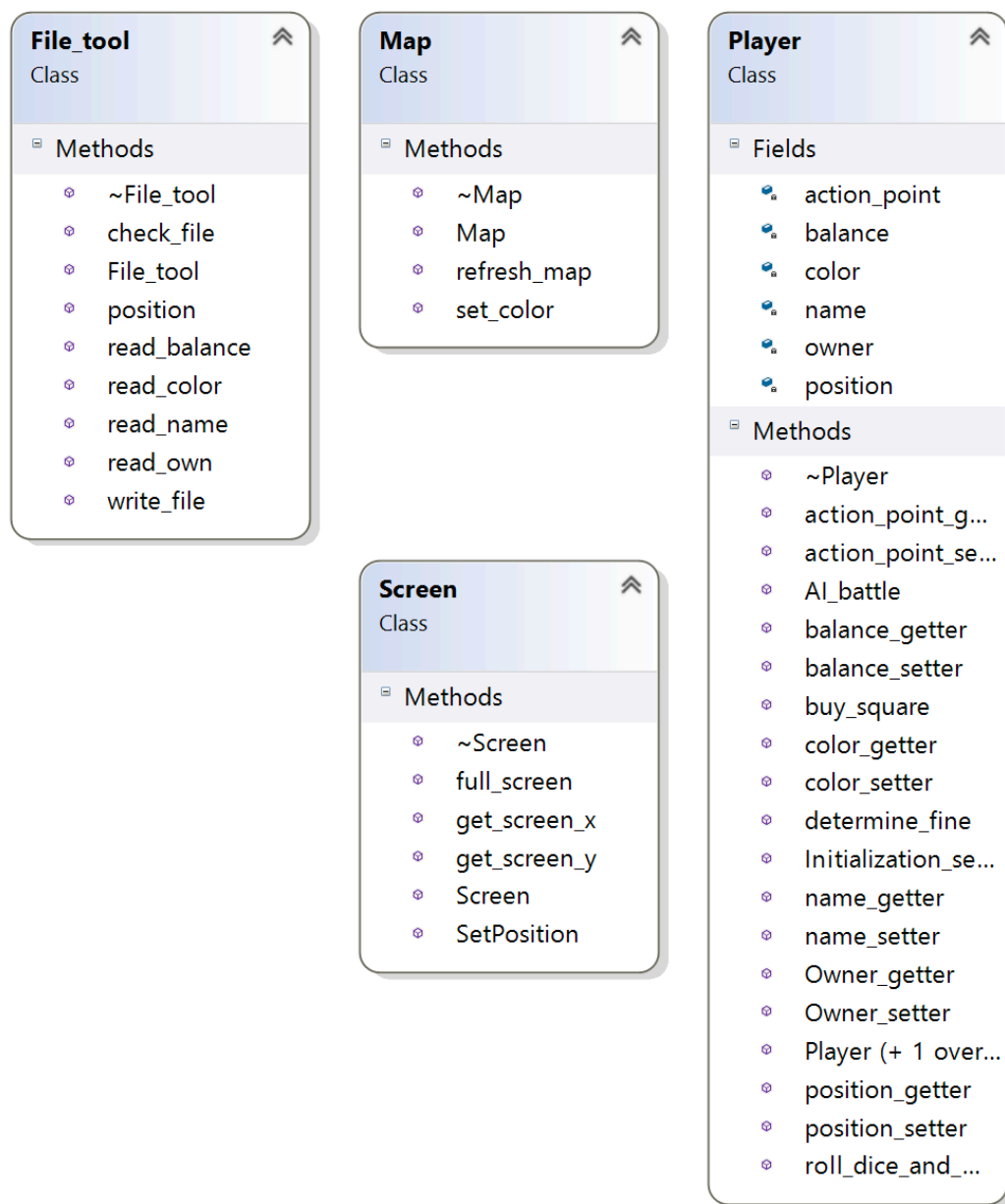


Figure 1 : UML Diagram

4.1.2 Player

In this class, I conceived some necessary parameters in order to facilitate the output of these values in subsequent work. The action points are used to store the points that players throw through the dice. Balance is also an important variable to measure the end of the game. In addition, I also set another variable called player color, which is mainly used to set the map to show the position occupied by the player. In addition, I used an array to display the property occupied by players and computers. If the property is occupied by players and is just a common investment, the land will be marked as 1, and if the land is doubled by players, the land will be marked as 2. Computer players are marked 3 and 4, respectively. In addition, if the plot is not occupied, the number is recorded as 0. Finally, the player's position will be recorded.

The following will introduce the specific methods and implementations of the Player class. All parameters are equipped with Getter and Setter for encapsulation. Therefore, the following methods are not specified:

action_point_setter/getter

balance_getter/setter

color_getter/setter

name_getter/setter

position_getter/setter

But there's one particular Getter and Setter -- Own. This is because Own is an array relationship, we may have some difficulties in setting values and getting values, and need to use pointers to complete. This is because the essence of arrays is actually pointers, so in order to avoid this problem, I use a simple way to set values directly through loops. Of course, there are address copies to achieve mutual assignment between pointers, such as cpy, but considering the cost of understanding, I use the following simple method directly.

```
void Player::Owner_setter(int set_owner[38]){
```

```
    for (int i = 0; i < 38; i++)
```

```
    {
```

```
        owner[i] = set_owner[i];
```

```
    }
```

}

In addition, it is noteworthy that it is quite difficult to get the whole array without using the number of pointers, so I use the following code to get the pointer, and then read the data in the main function to achieve.

```
int* Player::Owner_getter(){  
    int *Owner = new int[38];  
    Owner = owner;  
    return Owner;  
}
```

The return value of this function is a pointer. Because C++ 11 has its own method of deleting objects, it is not necessary to write deletion in the main function.

Now let's discuss the purchase method. The code for this method is as follows:

```
bool Player::buy_square(Player other, int square_price[], int type){  
    if (owner[position] == 0 && other.owner[position] == 0 && position != 0  
        && position != 19 && type == 1)  
    {  
        if (balance >= square_price[position])  
        {  
            owner[position] = 1;  
        }  
    }  
}
```

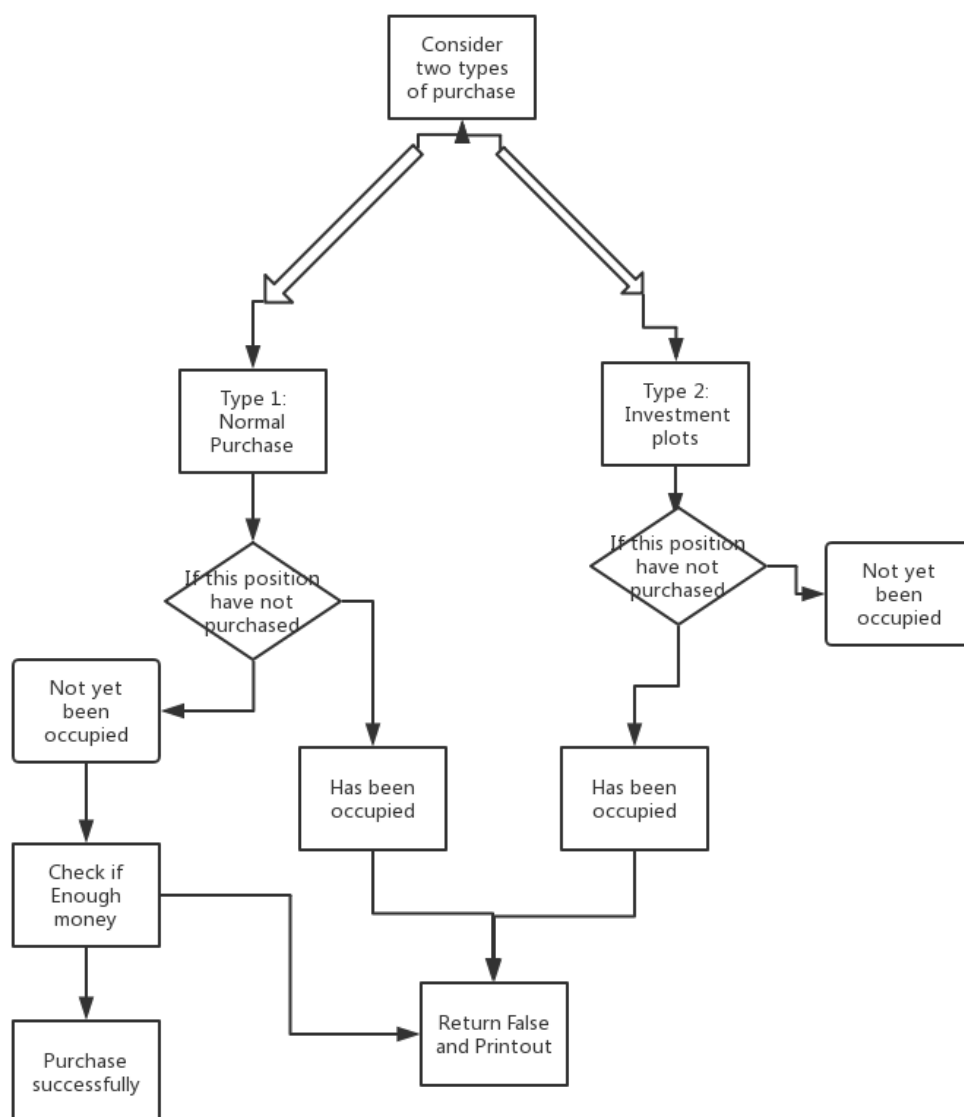
```
        balance = balance - square_price[position];
        cout << name << " Purchase Success in Number " << position
        << " and cost " << square_price[position]<<"$"<< endl;
        return true;
    }
    else
    {
        cout << name << " has No enough Money" << endl;
        return false;
    }
}

else if (owner[position] == 0 && other.owner[position] == 0 && position !
= 0 && position != 19 && type == 2)
{
    if (balance >= 1.5*square_price[position])
    {
        owner[position] = 2;
        balance = balance - 1.5*square_price[position];
        cout << name << " Purchase Success and upgrate in Number "
        << position << " and cost " << 1.5*square_price[position] << "$" << endl;
        return true;
    }
    else
    {
        cout << name<<" has No enough Money" << endl;
        return false;
    }
}
else
{
    cout << name << " has Purchased Failure in Number " << position
    << endl;
    return false;
}
```

}

}

The flow chart of this method is shown in the figure.

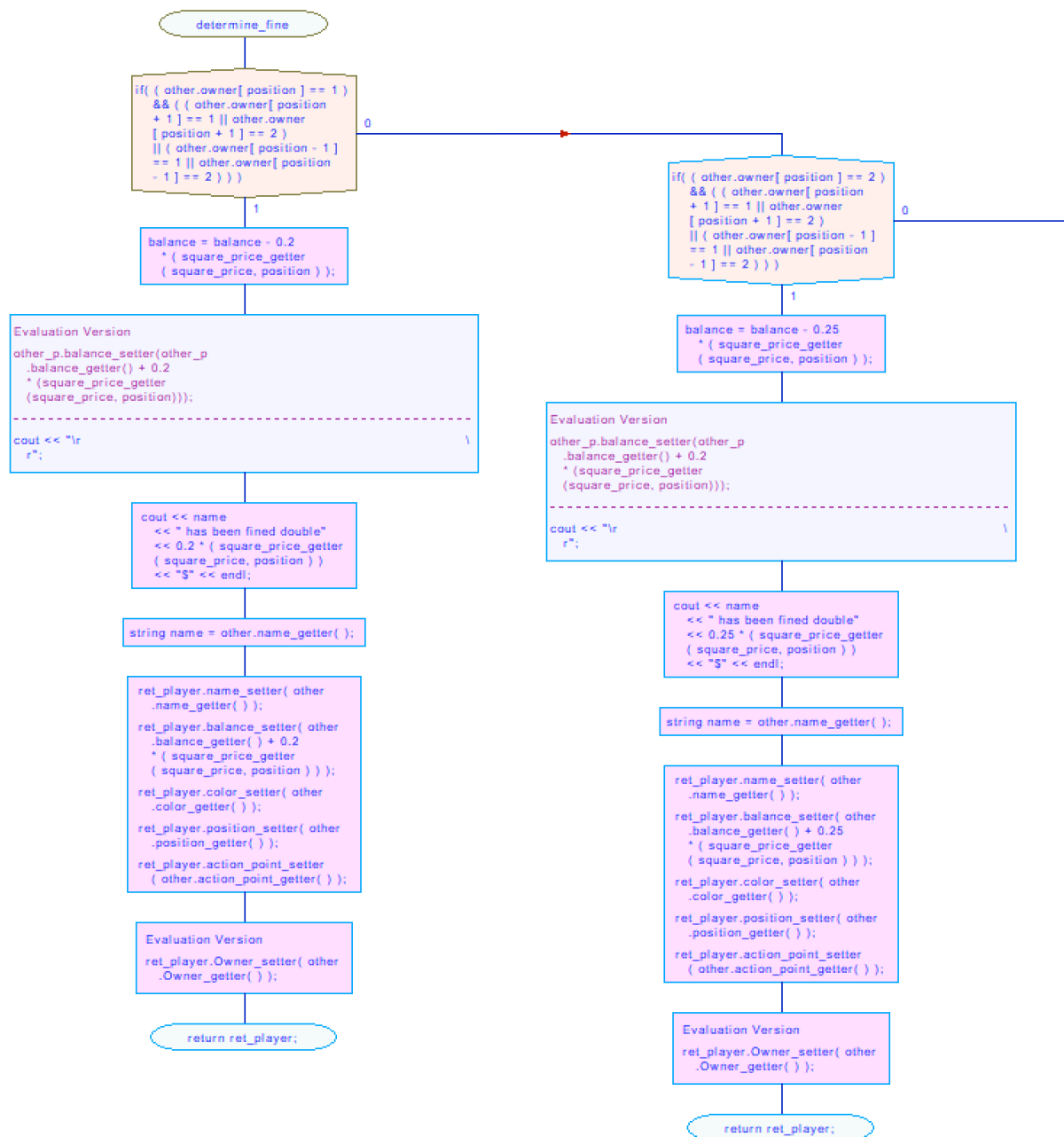


Next, consider the method of detecting whether a fine is imposed. Because before that, we introduced an array, which can provide us with the data we have.

Considering that a method can not return multiple return values, pointers should be used here, but the memory of this program is not too large, so the return value of this method is a class called Player.

Player Player::determine_fine(Player other, int square_price[])

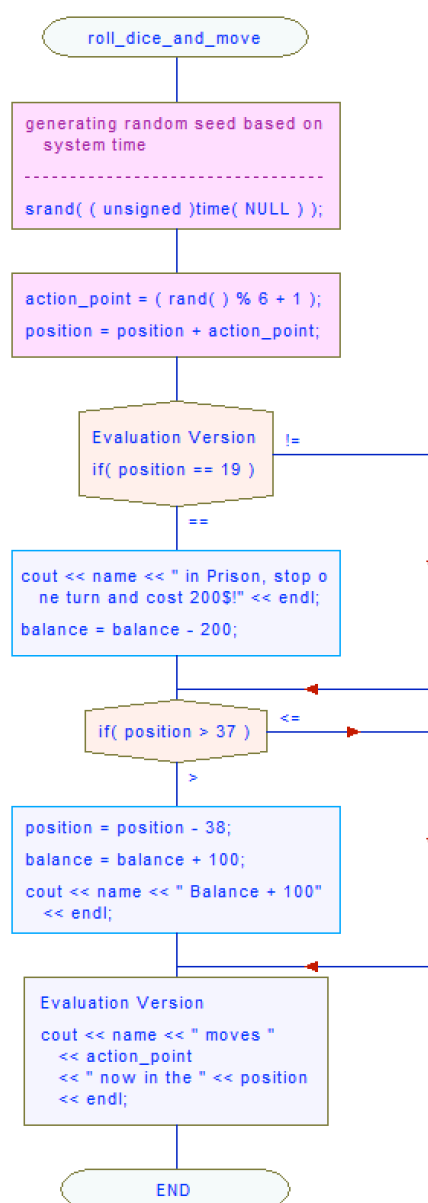
Then, when it detects that the player's location is exactly the land occupied by other players, the program divides the situation into four categories: the player stands on the land occupied by others, but the land beside the player is not occupied, so at this time, we will fine the land 10 percentage points. In addition, if someone next to the player is occupied by another player in a piece, the program will deduct the player's 20 percent fine. Similarly, since the player's location may be upgraded, there are two other possibilities based on the above discussion. So there are four possibilities. **The flow chart of the algorithm is as follows:**



In addition, there is another way to specifically control the way players throw dice.

This method generates a random number seed and simulates the player's steps in

this method. **The flow chart of the algorithm is as follows:**



Finally, the program reserved a computer and human game method, but did not give the function specific implementation code, because it is unreliable only by allowing the computer to buy with a certain probability, the future can make articles on this method.

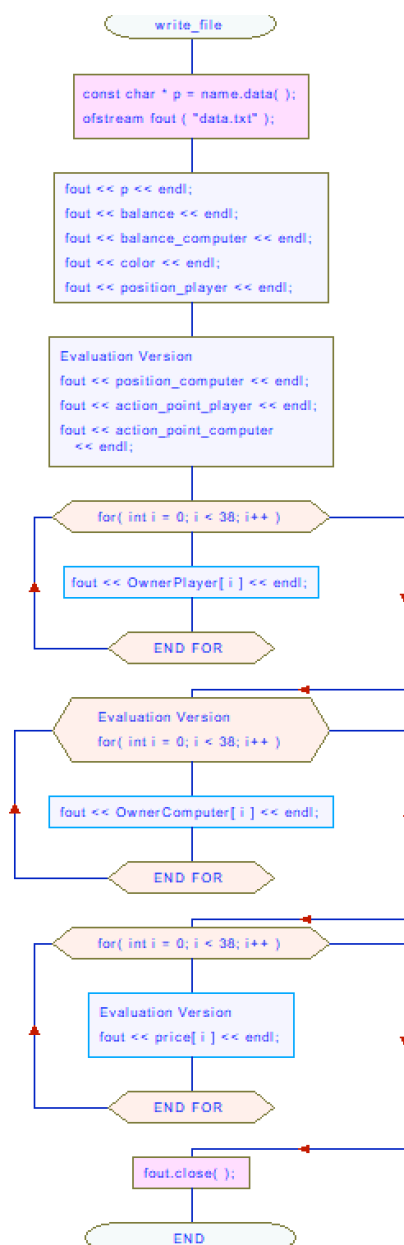
4.1.3 File_Tool

The purpose of this class is to use the class that the program will drop if the player wants to quit in the middle of the run. Because data needs to be persistent, data storage is a problem. Then, through this class, data can be read and written to the TXT document. This report will describe in detail the specific implementation code in this class.

First of all, considering the integrity of the file, if the loss of the file that saves the data may cause unexpected results, then the integrity of the file detection is also a necessary method. If the file does not exist, a method called `check_file` returns a False value. Finally, if the player chooses to continue the game, the player will be informed of the failure to save, and the player will be told the necessary reasons for the error. In addition, if not the whole file, but part of the file content is lost, so the

method *int File_tool:: CountLines (string filename)* will detect the number of rows of data, because the structure of the data has been determined, so the number of rows will not change, if there is a problem, the program will inform the user of failure to read the archive.

The method of file integrity detection is described above, and the method of reading game data is discussed below. First, the game data is stored in a method named *void File_tool:: write_file (string name, int balance, int balance_computer, int color, int position_player, int position_computer, int action_point_player, int action_point_computer, int Owner Player [38], int Owner Computer [38], int price [38])*. This method stores information about players and computers in a folder called data. Since the code takes advantage of file streaming, file writing can be written line by line. For an array, the storage method needs only a slight change, adding a loop to the code, and then the loop has a parameter that changes constantly, so as to realize the storage function of the file. **The flow chart of the method is shown in the figure:**



Finally, reading files is the most difficult problem, so it is necessary to write a line-by-line reading method in this method. A method is called *string File_tool::readTxt(string filename, int line)*. This method realizes reading files by line. It only needs to give the name of the file and the number of lines needed to

read the file. This method reads all the rows of the file, and then adds the data of these rows to a dynamic array of vectors, through which any number of rows needed can be obtained. However, there is a problem with this method. Every time a line is read, the program will traverse the entire file content once, which will consume a lot of computer performance. So we can use a common array to store these data, and then read and operate the data directly.

4.1.4 Map

Map Class is a special method to construct images. Firstly, this method interacts with Screen method, because the game needs to adapt the resolution of different monitors. This is what Screen class does, so building an appropriate method on the console allows the position of the image to vary with the resolution. Firstly, we will introduce the construction method of map method. This method uses HDC in Microsoft library, which is commonly called brush tool. This method allows users to draw on the console. I think the overall layout of the millionaire is good.

$RECT\ rect = \{ x / 20 + i, y / 20 + j, x / 20 + 50 + i, y / 20 + 50 + j \};$

This method can construct rectangles in different positions, and even set different colors.

In addition, considering that a part of the land needs to be clearly marked when the player takes possession of it, it is necessary to write a painting method. The name of this method is `set_color`, which allows you to set the color of a block at any location on the game interface. In fact, images and images are overlaid each other, which can eliminate the influence of the past. There will be a problem. If the graphics content is not pushed out of memory in time after building graphics, it may cause a certain memory leak, which is more dangerous to the impact of the program. However, painting tools do not have a suitable cleaning method. It is dangerous to use only the system ("`cls`") method. Although this method clears the content on the screen, the image will have some impact on the console.

Specifically, if the console is operated, such as dragging the console, the previously generated image will disappear. So it is necessary to use painting method here. But if you paint too many times in a console, it means an extreme value. For example, after more than 300 rounds, the image display will collapse. As a result, the program will not display any images at last. In order to solve this problem, in the program, I greatly reduced the number of image refresh, which makes the player alive to see the probability of image crash 0.

4.1.5 Screen

For Screen classes, the main function is to obtain screen resolution. So there is no default constructor for this class, but there is a method called `full_screen` for this class. This method first obtains the handle:

```
HWND hwnd = GetForegroundWindow();
```

Then the method of obtaining screen length and width is used in Screen:

```
int Screen::get_screen_x()
```

```
int Screen::get_screen_y()
```

Then get the window information:

```
LONG l_WinStyle = GetWindowLong(hwnd, GWL_STYLE);
```

Finally, set up console information:

```
SetWindowLong(hwnd, GWL_STYLE, (l_WinStyle | WS_POPUP |  
WS_MAXIMIZE));
```

```
SetWindowPos(hwnd, HWND_TOP, 0, 0, x, y, 0);
```


In addition, another key method of this class is to set the position of the specific text output in the console:

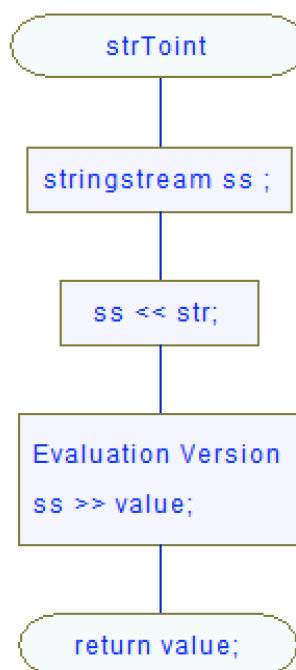
```
void Screen::SetPosition(int x, int y)
```

The specific implementation of this method is to query Microsoft Library, so the specific content is not described here.

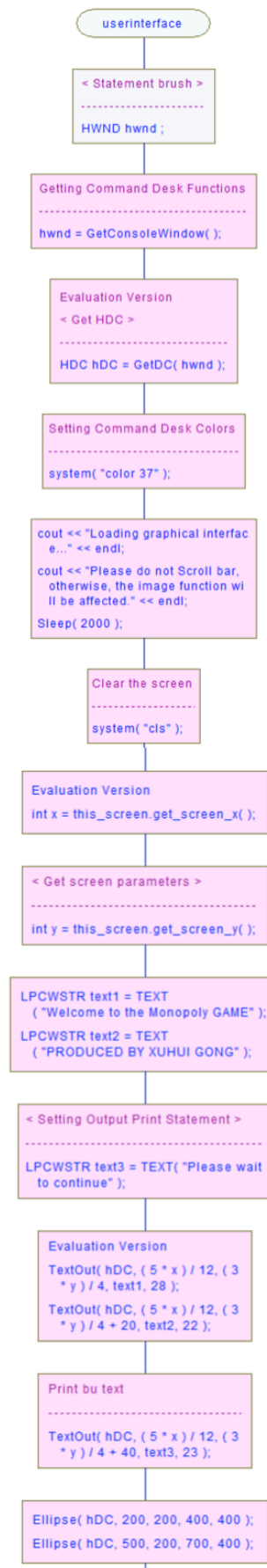
4.2 Main Function

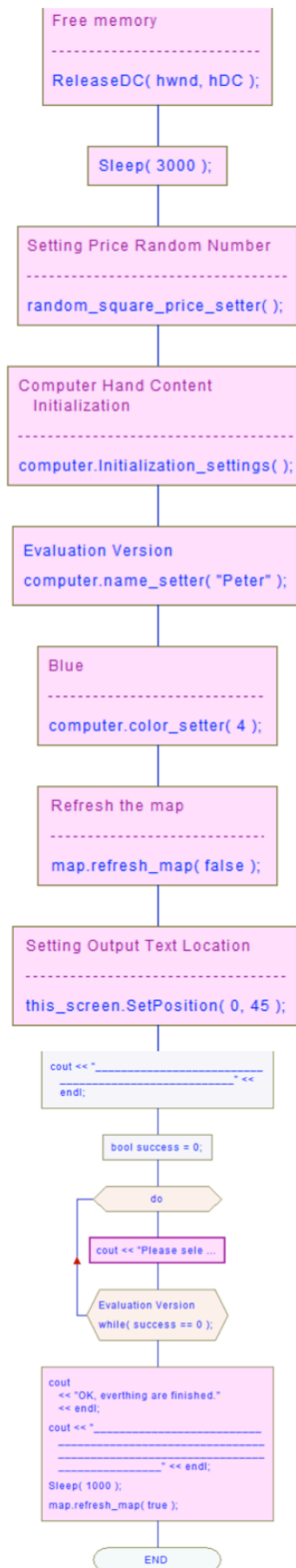
In the main function, two instance objects of the Player class are declared in advance: Player and Computer. This makes it easier to get or change the values of these two variables throughout the function. So other classes Screen and File_tool declare their public variables in advance. Because String is converted to int in file recovery, the declaration of this method is also reflected in the main function. The flow chart of this method is shown in the figure:

Declaration in main function

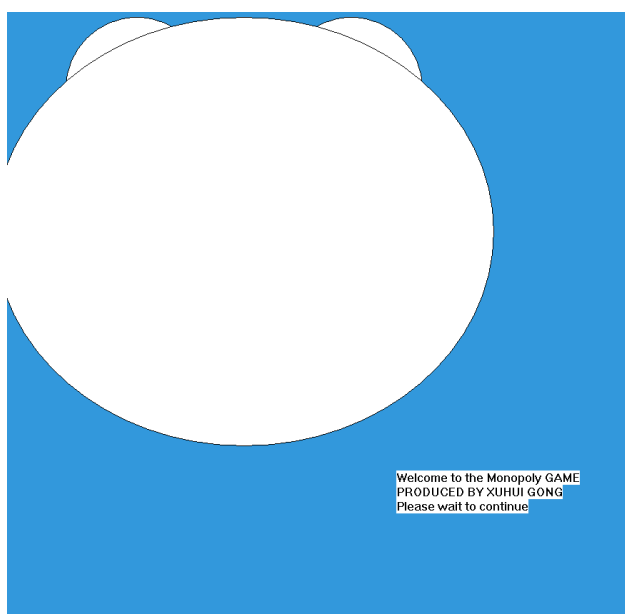


The user interface is discussed below. The main function of the user interface is to interact with the user. Therefore, in the user interface function, the user is required to input some choices, but the user is required to input a number, but the user enters a letter, which may lead to errors in the program, so it is necessary to write a method to detect the user's erroneous input parameters through the code. The flow chart of the method is described below.

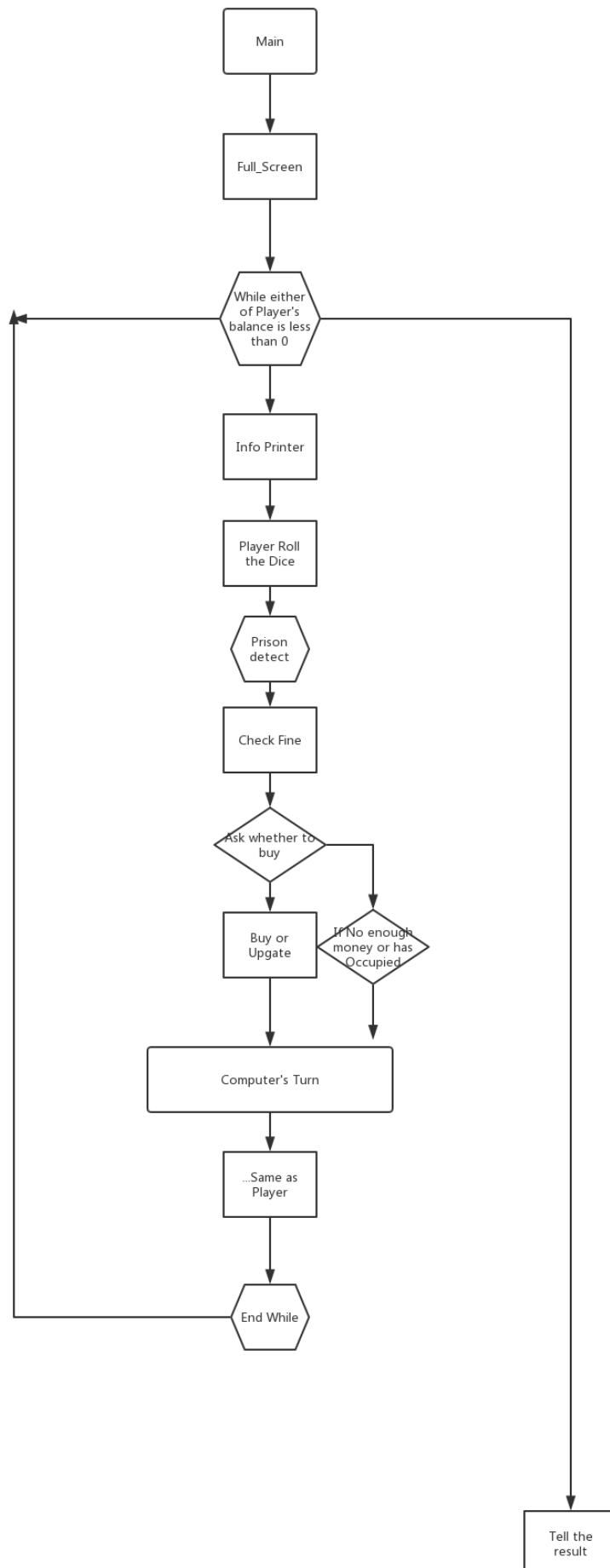




In the first half of the method, some images are displayed. Its purpose is to draw a startup picture for the game. The effect of this painting is as shown in the figure.



In the main function section, we first introduce the method of user interface, and then print some game information, such as player's identity, etc. Then go into the loop and know that the amount of money for any player is less than 0. In the loop, the player and the computer roll the dice separately and decide whether any player's position is in prison or not, if another player is in the prison. And in the loop, it detects whether the player is fined, which has been described in Player. The flow chart of the main function is shown in the figure.



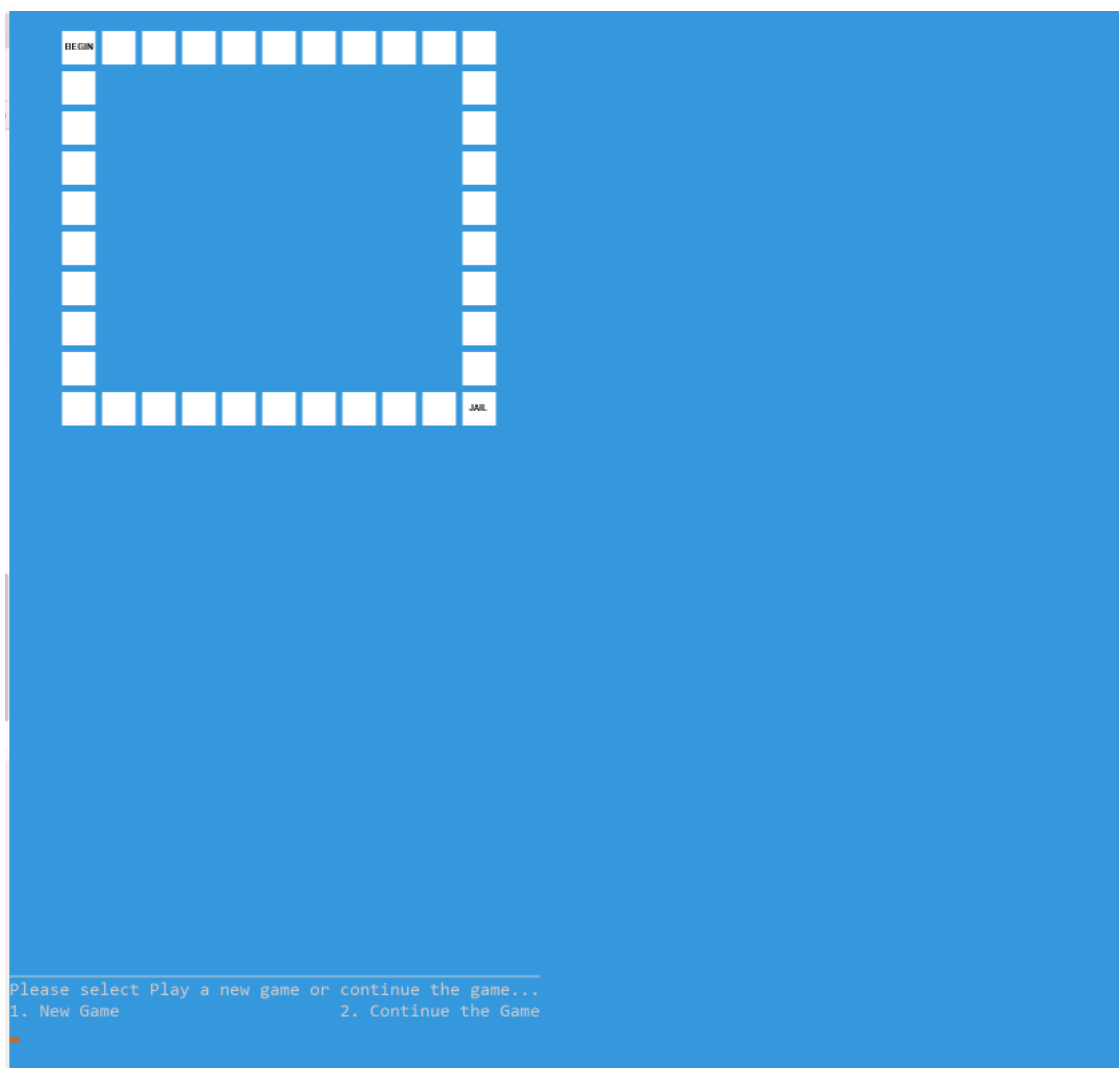
5. Implementation

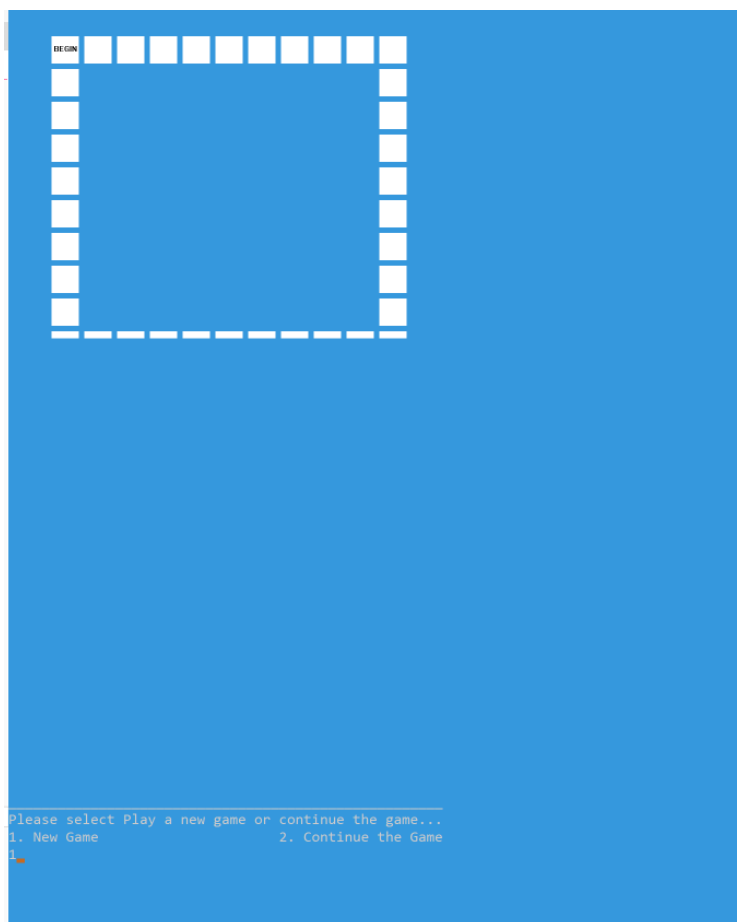
- *The algorithm of the part see as Assignment 3*

*Note: For Visual Studio, click **.solution** it is easy to run*

6. Testing and Verification

The first thing to test is the game creation function. First, the interface will have an option to enter a new game or restore the archive. Choose the first item here.



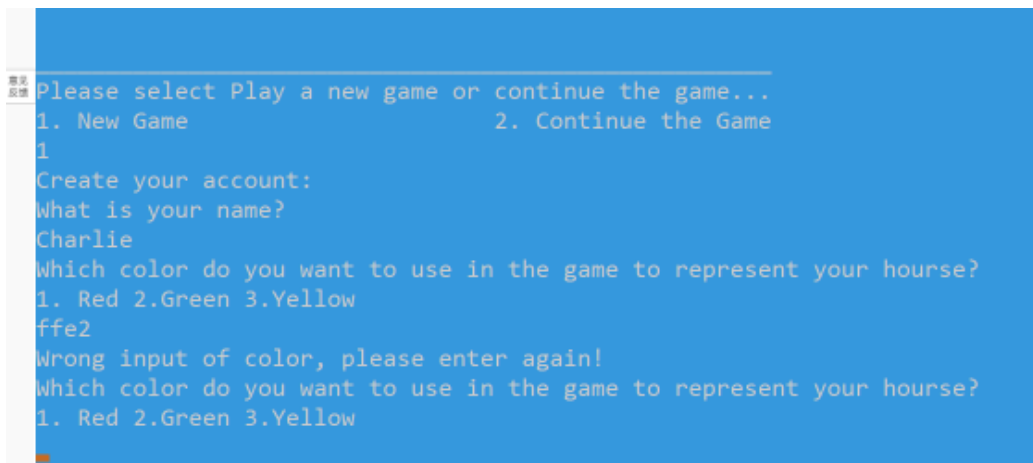


As you can see, image generation may fail due to changes in screen resolution, but don't worry, because the screen will refresh automatically after user input.

```
Please select Play a new game or continue the game...
1. New Game                                2. Continue the Game
1
Create your account:
What is your name?
Charlie
Which color do you want to use in the game to represent your hourse?
1. Red 2.Green 3.Yellow
2

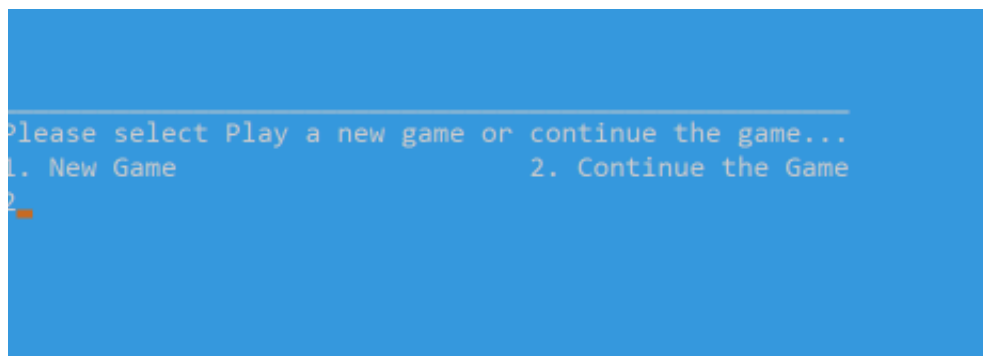
```

Here you can see that there is no problem with player input. Here, let's test what happens if players enter other illegal characters. You can see that the program correctly identified the input problem and jumped to the start to let the user re-enter.



```
Please select Play a new game or continue the game...
1. New Game          2. Continue the Game
1
Create your account:
What is your name?
Charlie
Which color do you want to use in the game to represent your hourse?
1. Red 2.Green 3.Yellow
ffe2
Wrong input of color, please enter again!
Which color do you want to use in the game to represent your hourse?
1. Red 2.Green 3.Yellow
2
```

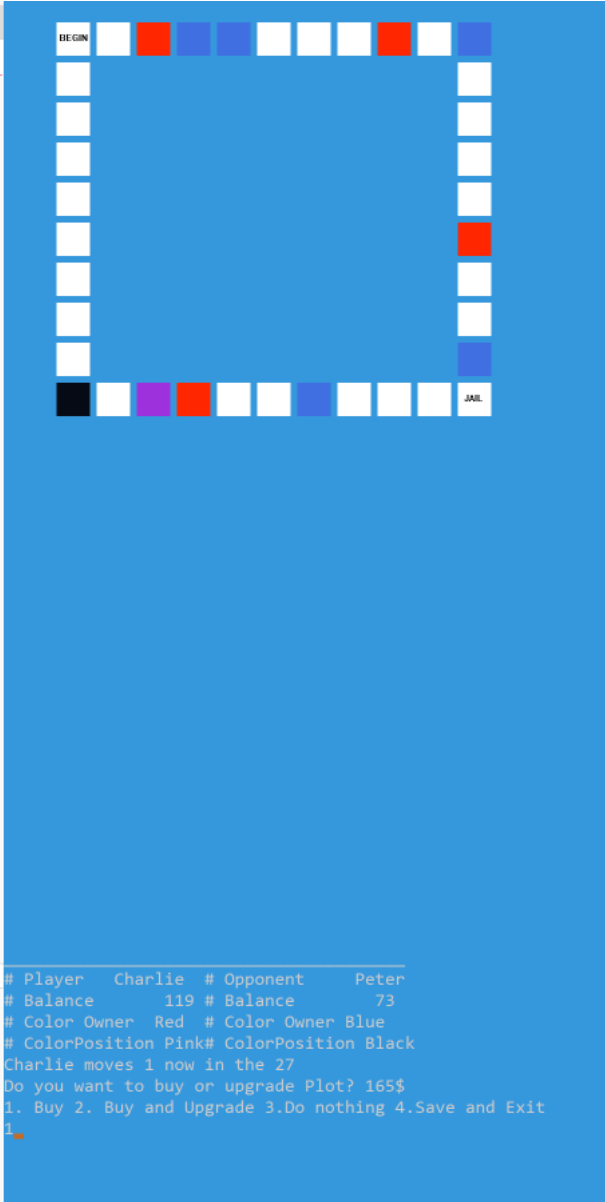
Subsequently, the data recovery function will be tested. First, the content of the document will be displayed.



```
Please select Play a new game or continue the game...
1. New Game          2. Continue the Game
2
```

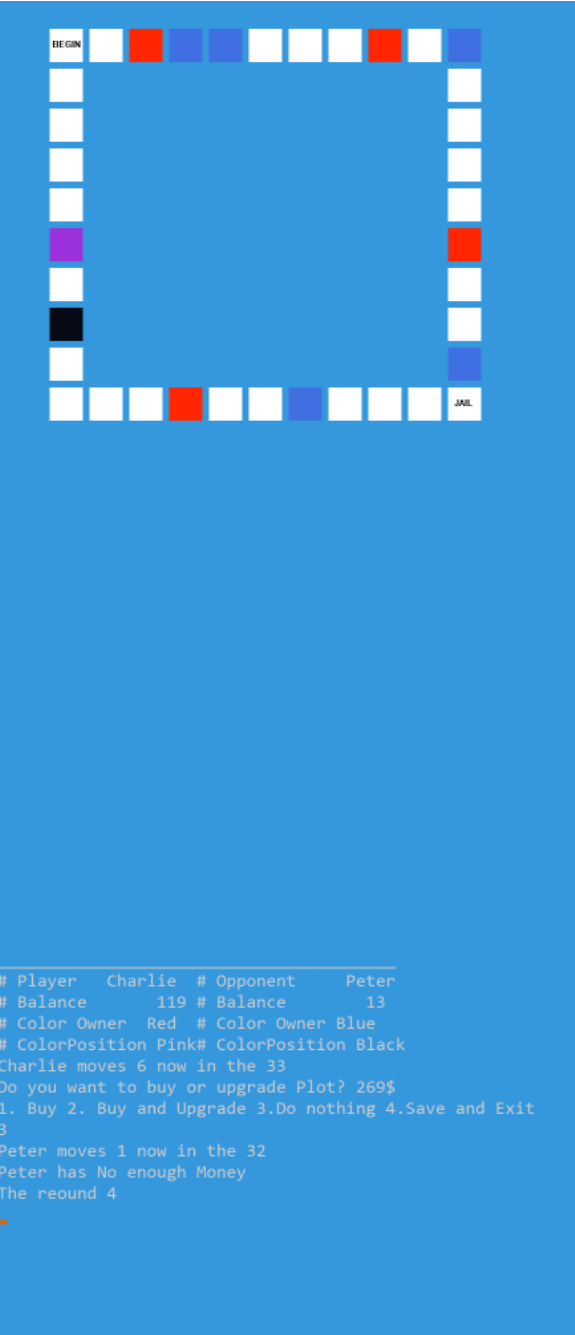
0

After that, the game functions will be tested a lot.



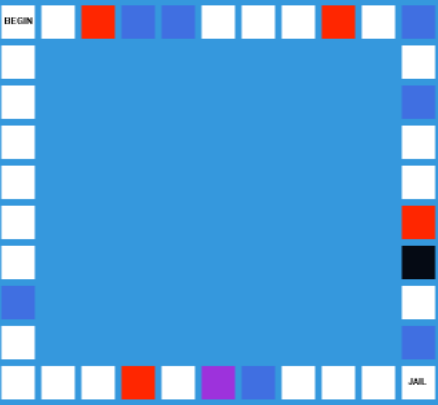
The screenshot shows a game board with a blue background and a white border. The board is 10x10. The top row is labeled 'BEGIN' and the bottom row is labeled 'JAIL'. The board contains several colored squares: a red square at (1,2), a blue square at (1,3), a white square at (1,4), a white square at (1,5), a white square at (1,6), a red square at (1,7), a white square at (1,8), a blue square at (1,9), a red square at (2,9), a black square at (9,1), a purple square at (9,2), a red square at (9,3), a white square at (9,4), a blue square at (9,5), a white square at (9,6), a white square at (9,7), a white square at (9,8), and a white square at (9,9). The terminal output at the bottom shows the game state and a prompt for Charlie to move.

```
# Player  Charlie # Opponent  Peter
# Balance      119 # Balance      73
# Color Owner  Red  # Color Owner  Blue
# ColorPosition Pink# ColorPosition Black
Charlie moves 1 now in the 27
Do you want to buy or upgrade Plot? 165$
1. Buy 2. Buy and Upgrade 3.Do nothing 4.Save and Exit
1
```

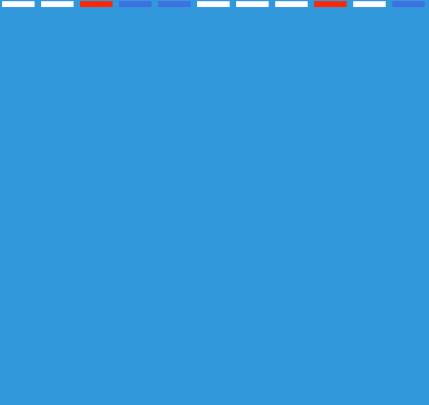


The screenshot shows the same game board as the first state, but with a different configuration of colored squares. The top row is labeled 'BEGIN' and the bottom row is labeled 'JAIL'. The board contains several colored squares: a red square at (1,2), a blue square at (1,3), a white square at (1,4), a white square at (1,5), a white square at (1,6), a red square at (1,7), a white square at (1,8), a blue square at (1,9), a red square at (2,9), a black square at (2,1), a purple square at (2,2), a red square at (2,3), a white square at (2,4), a blue square at (2,5), a white square at (2,6), a white square at (2,7), a white square at (2,8), and a white square at (2,9). The terminal output at the bottom shows the game state and a prompt for Charlie to move.

```
# Player  Charlie # Opponent  Peter
# Balance      119 # Balance     13
# Color Owner  Red  # Color Owner  Blue
# ColorPosition Pink# ColorPosition Black
Charlie moves 6 now in the 33
Do you want to buy or upgrade Plot? 269$
1. Buy 2. Buy and Upgrade 3.Do nothing 4.Save and Exit
3
Peter moves 1 now in the 32
Peter has No enough Money
The reound 4
```



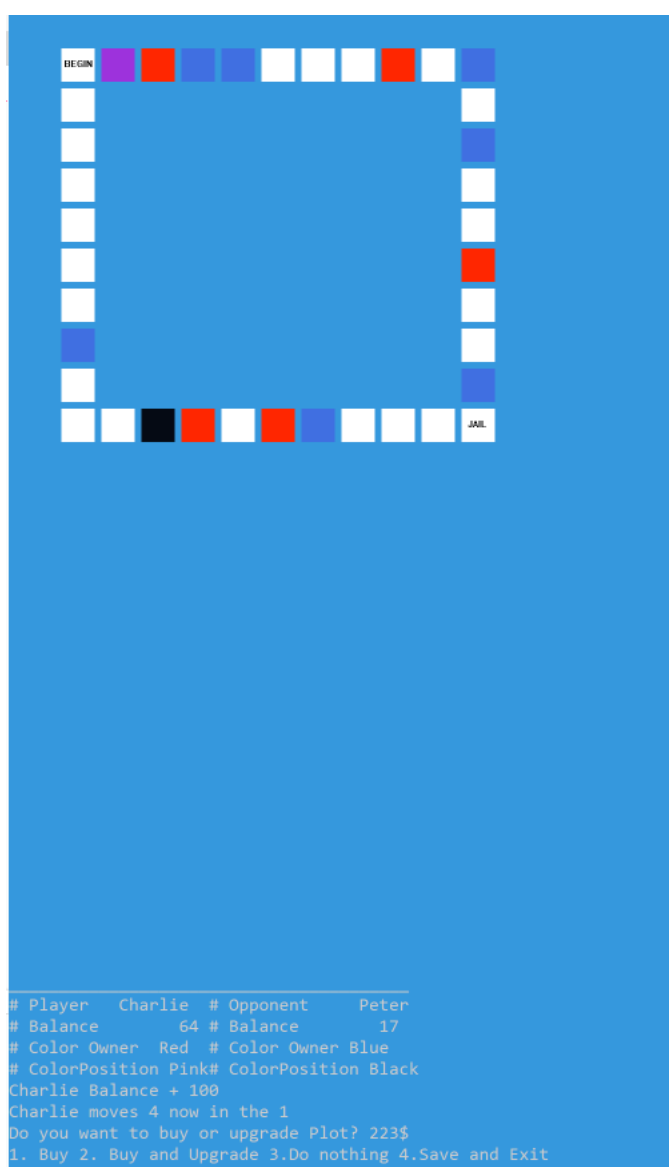
```
# Player  Charlie # Opponent  Peter
# Balance    182 # Balance    17
# Color Owner  Red # Color Owner Blue
# ColorPosition Pink# ColorPosition Black
Charlie moves 2 now in the 24
Do you want to buy or upgrade Plot? 118$
1. Buy 2. Buy and Upgrade 3.Do nothing 4.Save and Exit
1
Charlie Purchase Success in Number 24 and cost 118$
Peter moves 1 now in the 17
Peter has No enough Money
The reound 12
```



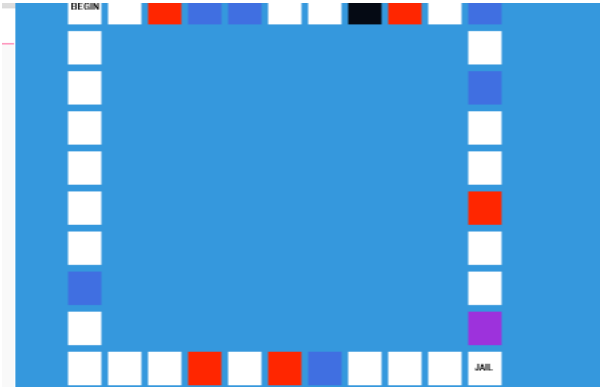
```
# Player  Charlie # Opponent  Peter
# Balance    64 # Balance    17
# Color Owner  Red # Color Owner Blue
# ColorPosition Pink# ColorPosition Black
Charlie moves 5 now in the 29
Do you want to buy or upgrade Plot? 108$
1. Buy 2. Buy and Upgrade 3.Do nothing 4.Save and Exit
3
Peter moves 1 now in the 18
Peter has Purchased Failure in Number 18
The reound 13
```

It can be seen that the last image is missing because the window is changed during the screenshot, so the image may disappear, but after this round the image will be refreshed.

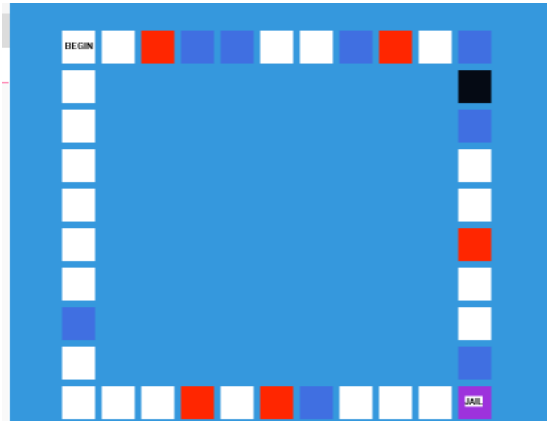
The following test players will increase their money by starting positions:



Fine function and prison function will also be tested.



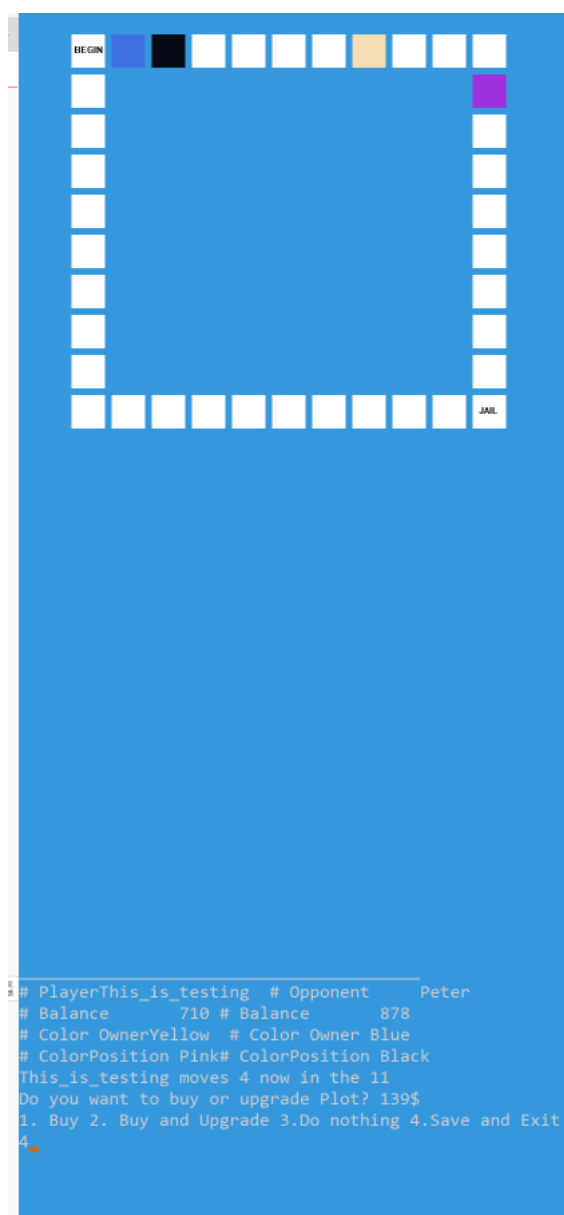
```
# Player  Charlie  # Opponent  Peter
# Balance    194  # Balance    62
# Color Owner  Red  # Color Owner Blue
# ColorPosition Pink# ColorPosition Black
Charlie moves 3 now in the 18
Charlie has been fined 24.7$
Peter earned 24.7$
Peter moves 3 now in the 10
Peter has Purchased Failure in Number 10
The reound 22
```




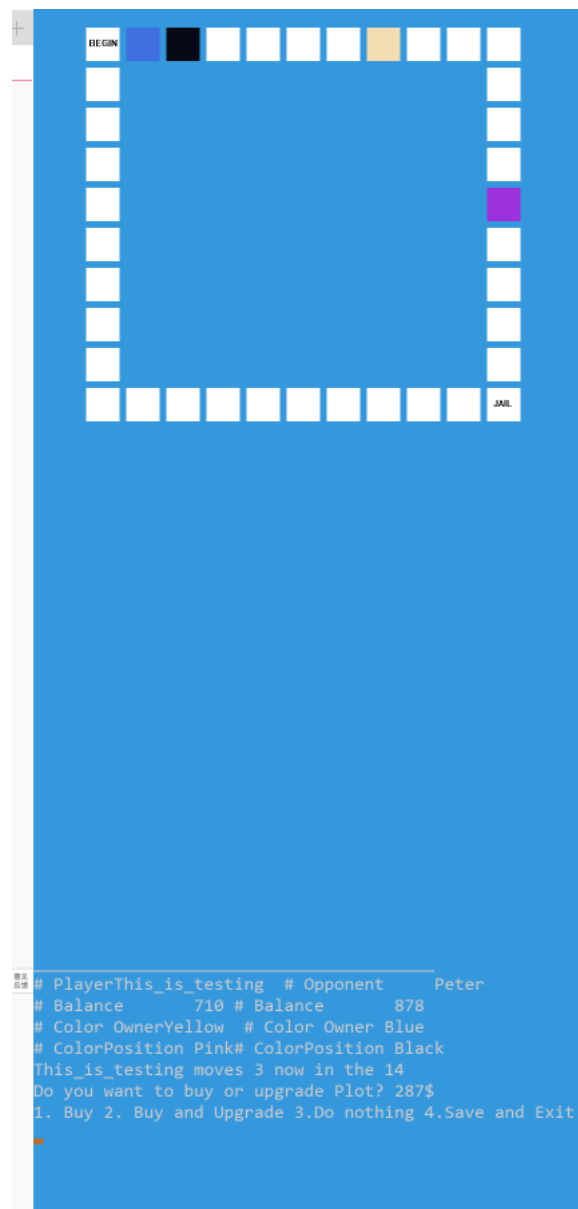
```
# Player  Charlie  # Opponent  Peter
# Balance    169  # Balance    86
# Color Owner  Red  # Color Owner Blue
# ColorPosition Pink# ColorPosition Black
Charlie in Prison, stop one turn and cost 200$!
Charlie moves 1 now in the 19
Peter moves 1 now in the 11
Peter has No enough Money
Peter moves 1 now in the 12
Peter has Purchased Failure in Number 12
The reound 23
GAME OVER
Press any key to continue . . .
```

Because I entered the prison, for the sake of game, I went to prison with a fine of 200 and waited for a round, so the player's amount was less than 200, and the game ended.

The game's storage is then tested. We create a new account and try to restore it.




```
Please select Play a new game or continue the game...
1. New Game                                     2. Continue the Game
2. 
```



7. Conclusion

This report fully describes all the steps that a program generates, whether it is from the beginning of the program, to the consideration of variables, to the choice of algorithms, and the writing of the program, to the final modification of the bugs.