# C Project Report: *plane fight*

## Abstract

We want to make a game about planes fight. In the game, there is a player. The player will control our plane to fire at hostile planes and at the same time avoid being hit by hostile planes. The game is not too easy and not too difficult. In a word, it is at intermediate difficult level. The player control the plane by pressing “w”, “a”, “s” and “d” to control our plane to move in all directions. And to fire at hostile planes by pressing “j”. The game has two modes. One mode is timekeeping mode. In this mode, the player is supposed to get higher score in limited time. And the other mode is boss mode. In this mode, the player is supposed to eliminate all the hostile planes. And then, there will be a boss emerging. The player is supposed to fight with the hostile boss and defeat it to win the game.

## Introduction/Problem Statement

Like the name, our project’s main function is about our plane fighting with hostile plane. Our plane fight game is a game that using the keyboard to control our plane to fire hostile planes. Our plane is on the bottom of the screen and hostile planes emerge from the top of the screen and slowly move towards our plane. We can move our plane and fire to hostile planes. And the hostile planes can also fire to our plane. There are two modes in our game, the first mode is survival mode in which we can fight with boss and the next mode is timekeeping mode. We choice Croatian Rhapsody as the background sound of the game, that is really suitable for our game . We can challenge the history highest score.

Our project has many advantages. Firstly our game has two modes, which can give players more and better game experience. What is more, the game is simple and easy to play. We use easyx to emerge our plane and hostile planes and it overcome the problem of screen dodging in some games.

To show the planes, we should use easyx function to draw the planes. To control planes, we should use a function to check if there is input on the keyboard and convert the input to the change of corresponding parameters. To check if the planes are hit, we should calculate the corresponding parameters by some formulas. To update planes’ location, we should make a loop and draw planes in every circle.

## **Group Division**

|  |  |  |
| --- | --- | --- |
| Name | student ID | College |

1.person one

Design how the game loop runs. Control the game process. Use the pointers to save all the parameters. Link functions related to the game loop. Make separate functions linked by passing parameters. Write startup function, show function, update with input function, update without input function and check function.

2.person two

Design how the main function runs. Link the functions in main function. Draw planes. Write hide cursor function, xy location function, lose and win animation function, my plane and hostile plane function, randomly showing planes function and bullet function. Write codes related to the file and saving score.

**3.**person three

Design boss section. Set background music. Draw boss plane and bullet. Link functions of boss and control the boss process. Write boot animation function, boss show function, boss update with input function, boss update without input function, boss check function, boss plane function and boss bullet function.

## Analysis

First of all, we are supposed to show planes on the screen. At first, we wanted to use “printf” function to print symbols such as “+”, or “\*”, and combine them to show planes. Then we also thought up that we can use “easyx” function to draw lines and paint color to draw planes. In the beginning we thought that to print planes was easier but to draw planes was more beautiful. Eventually we chose to use “easyx” function to draw planes.

To decide which way we are going to use, we wrote codes in two ways and execute these two showing programs then we got two results. We compared these two results. We found that if we chose to print “+” or “\*”, the plane would twinkle and when we move the plane, it became not fluent. However, when we use “easyx” function to draw the plane, it ran fluently. And the picture surely looked nicer than a pile of symbol.

This is our experiment before programing. And we get the result. The symbol planes are like this:



And the bullet is like this:

C:\Users\Administrator\AppData\Roaming\Tencent\Users\2848123049\QQ\WinTemp\RichOle\H3L3$}P]PR_0J{RSOVJF$R5.png

However, if we use easyx function to draw the planes, the picture will like this:



And the bullet is like this:

C:\Users\Administrator\AppData\Roaming\Tencent\Users\2848123049\QQ\WinTemp\RichOle\NEMEAIGKFBZ3QRHXQ3EJN}2.png

Obviously, the lines and different colors are more beautiful than symbols.

We thought the reason that the later one was more fluent than the former one was that computer run easyx function is faster than printf. So if we use printf to print planes, the screen might flash. Consequently, although the easyx function is a bit more difficult than the printf, we eventually chose to use easyx to draw planes. But as for the boot animation, we chose to use printf function to print the phrase “plane fight” by “\*”. Because we found that this way would have the best effect for its animation.

Secondly, we are supposed to move our plane. So we will check the input from the keyboard and convert the input to the movement of planes. When we press keys “w”, “a”, “s”, “d”, the plane will move forward, leftward, backward and rightward. And we design that when the key “j” is pressed, our plane will fire. A bullet will emerge from the top of the plane. And this bullet will move forward. Its speed is a lot faster than the plane. If the bullet hits hostile planes, they will disappear.

Thirdly, hostile planes will emerge at the top of the screen randomly. So we will use a random function to output a random number to be the parameter of the location of hostile planes. We will make hostile planes move forward. So we want to use a loop. In each loop we change the parameter of the location of planes. In this way we can make planes move.

We will also control the planes’ speed. Planes should not move too fast or too slow. The way to control the speed will be explicitly explained in the implement part. Hostile planes can also fire to us. So when we play this game, we will not only try to fire to hostile planes, but also pay attention to avoid to hostile planes’ bullets.

Fourthly, we will set two modes. The first mode is the timekeeping mode. In this mode, the player will have finite time to fire hostile planes. The more planes player defeats, the higher score the player will get. When the left time equals to 0, the loop will break and game is over. You now will have a score. Our game will record your score and compare it to the history highest score. If it is higher than the former score, your new score will take the place of the history highest score and become the newly history highest score.

In this part there are only ordinary hostile planes. Each plane is small in size and each plane only has one life. Once it is hit, it will disappear. Of course, their bullet is also small. Once your plane is hit, your life will only minor one. However, in the other mode, boss mode, first, you must kill 4 hostile planes, and then a boss will emerge. This boss is bigger in size than ordinary planes. And it has more lives than ordinary planes. Moreover, its bullets have bigger destruction. Once your plane is hit by boss’s bullet, your life will minor two. But the boss’s bigger size means that you will have more chance to hit the boss.

In each mode, if you accomplish the mission, and you survive, the screen will appear “win!”. If your life equals to zero, the screen will appear “lose”. After playing the game, you can choose to exit the game or to play the game and choose the mode again.

The above context is about the main analysis of our program. And the next part is about some other analysis.

We want to limit our plane in a finite part of space. So do hostile planes. And if bullets fly out of this part of space, the bullets will disappear. To do this, we will control parameters which represent the location. If some parameter is bigger than some number, this parameter will be change to make sure the location that the parameter represents is in the limited space.

We want to set a background music called “Croatian Rhapsody”, and play the music again and again until the game is over.

There are three planes on the screen. At first, three planes will emerge at the top of the screen simultaneously. If one plane is hit, another plane will emerge at the top of the screen and this will make sure that there are three planes on the screen.

If one hostile plane is flying out of our limited space, then, another plane will emerge at the top of the screen.

To increase the difficulties of the game, we decide that the bullet must hit precisely at the center of the hostile plane. However, this also decreases the difficulties. Because hostile planes’ bullet also must hit precisely.

And the final part is about the function relative to the game effect.

We decide to write several separate functions to implement each function. And we will link each function to a complete main function. We think that this design will be easy for us to write code and it is easy for computer to run the program.

Each function will be linked by parameters.

And as for the pointers, at first, we think that if we use the global parameters, the program will be easy to write. Because if we use the global parameters, in each function, we just change the global parameters. And we do not have to pass all the parameters in our program. But professor Hao told us not to use global parameters because it is dangerous. So we finally use the pointers to save all the parameters and the work is more than using the global parameters.

In the function about showing the planes, we should have some codes to check if the planes should be showed. And this make us to use parameters to check if the value is bigger or smaller than 0.

As for some detailed information about the easyx function, we will draw some lines and to fill the space between the lines with different colors. And then the picture will look like a plane.

So after all the analysis, we have had the designs and the problems than we will pay attention to in mind. And certainly, in the programing process we met a lot of problems that we did not think about. At this time, we found the answer from the internet or asked professor Yibo to solve the problems. And in this process we better or design again and again. And the next part about design is the final version.

## Design

First of all, the program is composed by many separate functions, so we will write all the separate functions and link all the functions in a loop. Each function will be explicitly explained in implementation part.

To accomplish our program access, in our main function, first, we will initialize all the parameters. And these parameters will make sure that all the planes will appear at the proper location at the beginning of the game. Then, after executing the boot animation and beginning to play the background music, we will ask the player to choose mode. And then there will be two separate loops. If player chooses mode one, then the player should input the number “1”. If player chooses mode 2, he should press the key “2” and press “enter”. And our program will check the input and go into the according loop.

In the first loop which is related to the timekeeping mode, we will set a parameter called time. We will give time an initial value. First, we execute the show function. In this function, we draw planes according to their initial location parameters. Next, we will execute the function called update\_with\_input. In this function, we will check if there is input. If there is, the function will read the input and convert the input into the change of the according parameters. And our plane’s location will change according to the change of the parameters. If player presses “j” to fire, the parameter of bullet’s location will be given a value. Whether hostile planes disappear is also related to this function. Then, we will execute the function named update\_without\_input. In this function, the parameters of flying bullets will be added in order to make the flying bullet look like going forward. And the parameters of hostile planes’ location will also be added to make hostile planes go forward. The hostile planes will spontaneously fire to our plane. And that hostile planes will randomly appear at the top of the screen is also related to this function.

In the first loop, the next function is named check. In this function, all parameters will be checked in this function. We will use a lot of formulas to calculate the relative relationship of many parameters. We use these relationships to decide whether hostile planes’ bullets hit our plane and whether our planes’ bullets hit hostile planes. If the distance of the parameters of our plane’s location and the parameters of the hostile planes’ bullets is smaller than some number, we will think that the hostile planes’ bullet hit our plane and our life will minor one.

At this time, our first loop is almost done. The remained part is that we use the function “Sleep” to slow down the program to a proper speed. Then we let the parameter “time” minor one. Finally, we will check the parameters “time” and “chance”. (the chance represent the remained life.) If the time equals to 0 or the life equals to 0, we will break the loop and go into the next part. If both time and chance do not equal to 0, we will go back to the beginning of this loop and continue to execute the loop and check the parameters again, until one of the parameters equals to 0.

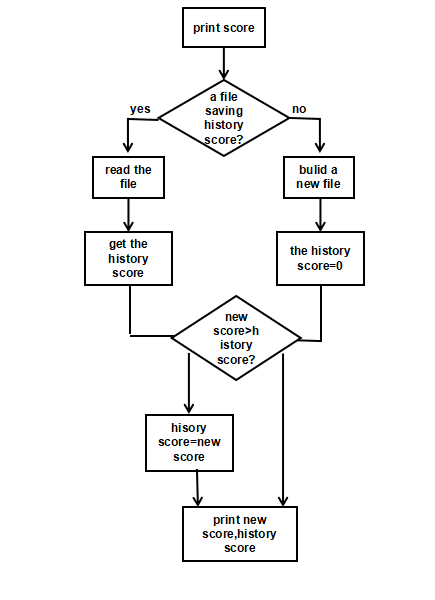
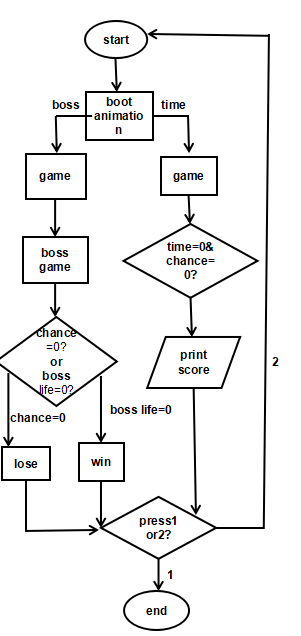
Once the loop is break, there will be a file built. There are two parameters “score” and “history\_score”. The value of the parameter “history\_score” will get from the file. There is a check. First, we will check if there is a file. If there is, we will read the file and get a number. We will give this number to the parameter “history\_score”. And then we will compare the parameter “score” and the parameter “history \_score”. If the score is bigger, will give the score’s value to history\_score. And if the score is smaller, there is no following code. Finally, we will print the score and the history highest score on the screen. By now, the first loop is end.

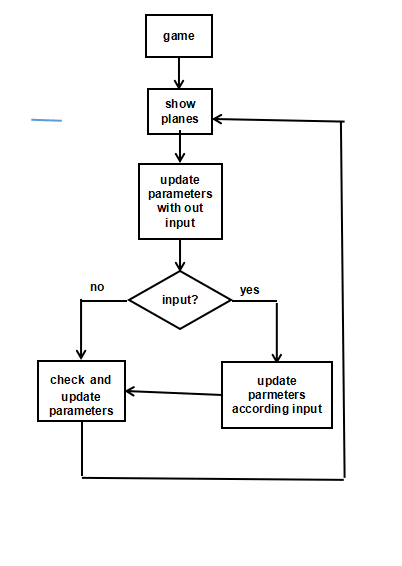
The second loop is related to the boss mode. This loop is similar to the first loop. But they also have some difference. First, this loop does not have the parameter “time”. And the condition for the break is also different. In this mode, we check the parameters “score” and “chance”. If the chance equals to zero, the loop will break and there will be a word “lose” printed on the screen. If the score equals to some number, we break the loop. Then we go into the boss part. In boss part, we check the “boss\_life” and the “chance”. If boss life equals to zero, this loop will break and there will be a word “win” printed. If the chance equals to zero, the word “lose” will be printed on the screen.

At the end of the main function, we will let player to input a number. If the player presses “1”, we will go back to the beginning of the main function and play the game again. If the player presses “2”, we will end the main function. And by now, all of this game is over.

Next, we will use a graph to show our design more clearly.

The first picture is about our main function. In the first picture, the game and the print score is composed with a lot of contains, so we write these two part in other pictures. These two pictures follow the main function picture.





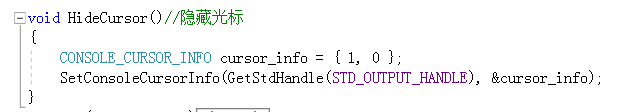
## Implementation

Firstly, we will explain all the separate functions.

1.HideCursor

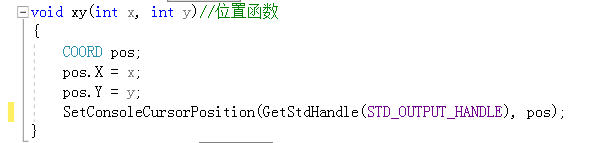
We hide the cursor to make the screen cleaner.

The following picture is about this function.



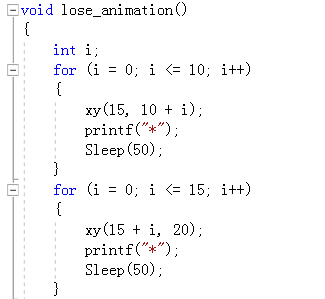
2. xy()

We use this function “xy” to go to the location and print letters and symbols at the location. The following picture is about this.



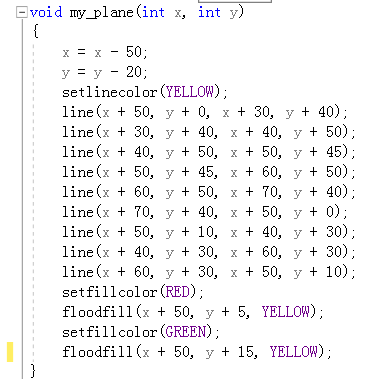
3. boot animation, lose animation, win animation

We use “boot\_animation”, “lose\_animation”, and“win\_animation” to print “plane fight”, “lose” and “win!”. These three functions are composed of many “for” loops These loops print many “\*”, and these “\*” compose all the words. The following picture is a part of our animation functions. There are totally 450 lines in our animation functions. The following picture is about this function.



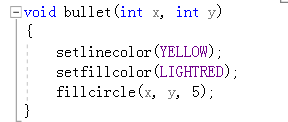
4. my plane, hostile plane

We use easyx function to draw all the planes. We first draw the lines. Then we fill the space between the lines with different colors. The following left picture is our codes about my plane. And hostile plane codes are in the same way. The right pictures are planes.

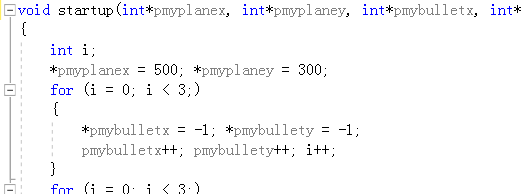
5. bullet

Our bullets are circles. So we also use easyx function to draw circles. The following left picture is about our bullet code. The right picture is our bullet.

 C:\Users\Administrator\AppData\Roaming\Tencent\Users\2848123049\QQ\WinTemp\RichOle\G]8%~%TT9Z)}PTMK]]9)C}S.png

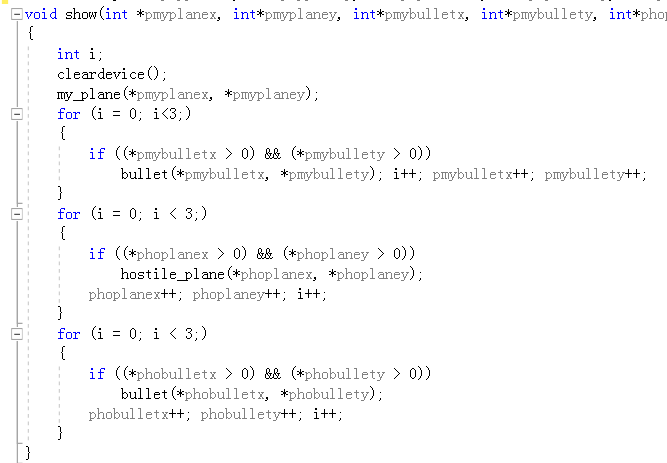
6.startup

In this function, we will initialize all the parameters. We use pointers to save all the parameters. So at the beginning of all the functions, we should pass all the pointers so that the program can recognize the pointers. There are too many pointers so we just show several pointers in the following picture. In the startup function, we let all the parameters which represent the hostile planes and bullets’ location equal to a minor number. And our plane location parameters equal to 500 and 300. We regulate in the show function that if the parameters are smaller than 0, we do not show the plane or the bullet. So during the first loop, there are no hostile planes. The following picture is about this function.



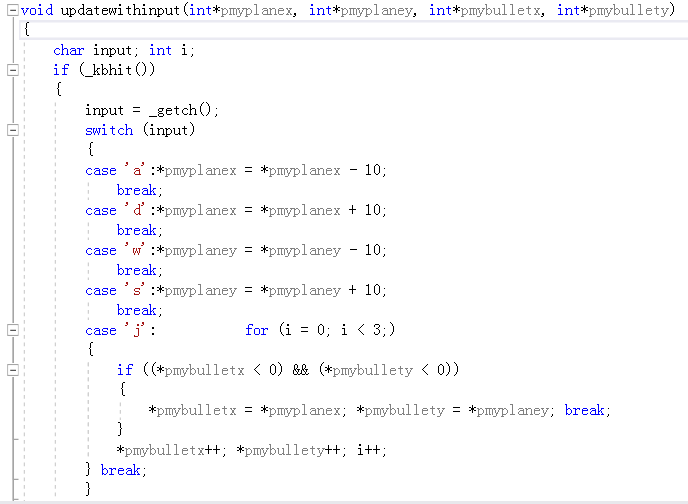
7.show

In show function, we use the my\_plane function, hostile\_plane function and bullet function. In show function, there is a “if” check. This means that only when the parameters is bigger than 0, we will show the planes and bullets. We pass the parameters to show function so that we can know all the planes and the bullets’ location according to the parameters. We regulate that there are three hostile planes in total. And we use “for” loop in this function to make sure that we can show all three function. The following picture is about this function.



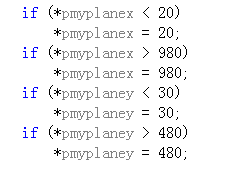
8. update with input

In this function, we mainly update parameters according to our input. First, we update our planes’ location parameters according the input on the keyboard. The following is about movement of our plane.



For example, if we press “a”, the parameter myplanex will minor 10, which means our plane move left. As for the “j” case, we will firstly check which bullet’s location is smaller than 0. We will choose this bullet and give our plane location to its location. This will appear that there is a bullet fired from our plane.

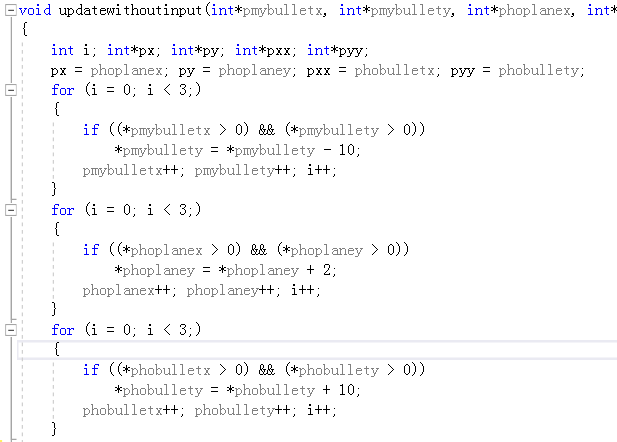
The next part is about our limited space. And the following picture is about this function.



If after an input our plane is out from our limited space, we will let our plane go back to the limited space. This will appear that when our plane is at the right edge of the limited space, there will be no effect to press “d” to let our plane to move right.

9. update without input

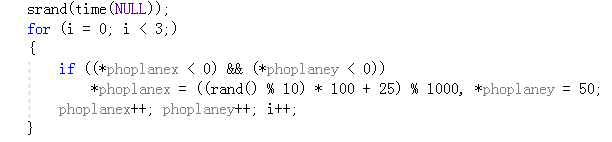
In this function, we will update many parameters without input. This means that even if there is no input, many parameters will change automatically. They are my bullet location, hostile plane location and hostile plane’s bullets’ location.



In every loop, the parameters will be updated once. For example, the hostile planes’ y axis parameters will plus 2 in every loop. So hostile planes will move forward. We just let the parameters plus 2 in each loop, which make sure that the plane will not move to fast or too slow. By using the same mechanism, hostile planes’ bullets as well as our bullets will move forward automatically.

Then we use a random function to randomly emerge hostile planes. When a hostile plane is hit, we make its location equals to a number that is smaller than 0. So in the random function we will check if the bullet location is small than 0. If it is, will give its location a random number bigger than 0 and smaller than 925. And this will appear that there is a plane randomly emerging from the top of the screen.

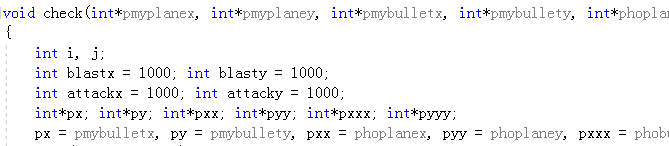
The following picture is about our random function.



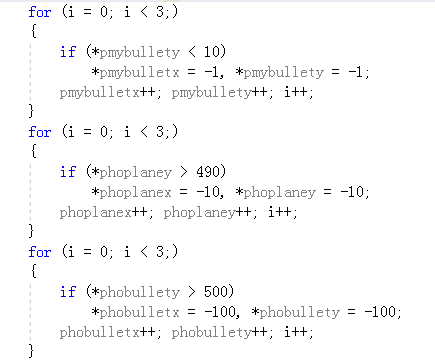
10.check

In the check function, we will use some formulas to check some parameters.

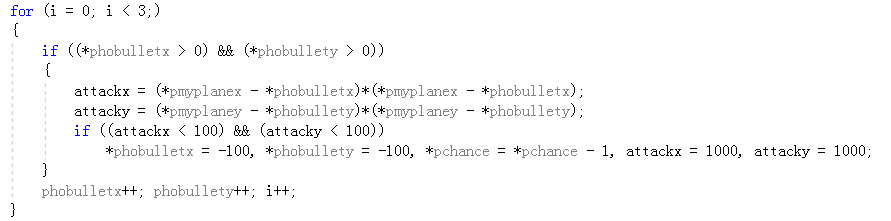
First, we give some parameters some values. The following picture is about this.



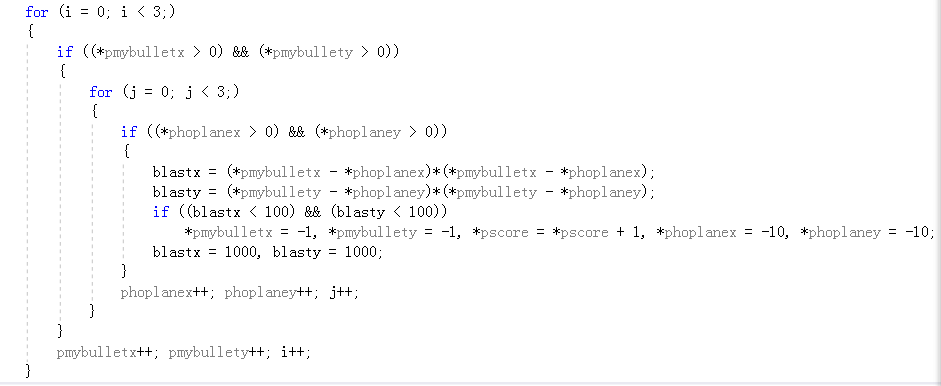
And then, we check my plane bullets’ location, hostile planes location and hostile planes’ bullets location. For example, if one of our plane’s bullets’ y axis parameter is smaller than 10, we will think that this bullet is outside our limited space and we will not show this bullet. So we let its y axis parameter equals to -1. If the parameter is smaller than 0, we will not show the bullet. The following picture is about this part.



In the next part of this function, we will check if planes are hit. So we will calculate the distance between planes and bullets. First, we calculate the distance between hostile planes’ bullets and our plane. We let the parameter “attackx” equals to the absolute value of the hostile planes’ bullets’ x axis parameter minor my plane’s x axis parameter. And We let the parameter “attacky” equals to the absolute value of the hostile planes’ bullets’ y axis parameter minor my plane’s y axis parameter. If the attackx is smaller than 100 and attacky is smaller than 100, we will think that the hostile planes’ bullets hit our plane. And we will let hostile plane’s bullets’ location equals to -100. This means that hostile planes’ bullets will not show in the next loop. And we let our plane’s life minor one. In the end, we let attackx and attacky equal to 1000, this make sure in the next loop there is no error. The following picture is about this part.



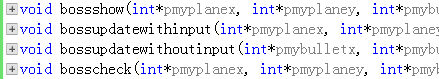
Then the following picture is about the other check.



We will calculate the distance between our planes’ bullets and hostile planes. And the mechanism is almost the same as the former one. The first difference is that there are three hostile planes and we will calculate for three times. The other difference is that if we hit one hostile plane, our score will plus one.

11.boss part

In boss part we also have several functions. They are “boss show”, “boss update with input”, “and boss update without input” and “boss check”. The following picture is about this.



And the mechanism in boss part is almost the same as the former part. But they also have some difference. The most obvious difference is that the parameters are completely different. The first other difference is that the boss show function is different. The boss figure is bigger than other planes. And so do boss’s bullets. And the following picture is the boss and its bullet. Except this, other mechanism is the same.

 C:\Users\Administrator\AppData\Roaming\Tencent\Users\2848123049\QQ\WinTemp\RichOle\OJOKN`BHNI1@RPAE~0[B8O2.png

As for the boss update with input function, all the mechanism is the same as the former corresponding one.

In the boss update without input function, there is a difference. The boss will not go forward, it will only move rightward or leftward randomly. So we write another random function in this part. We let the boss x axis parameter plus a random number. This number may be a positive number. But it may be a negative number. This means the boss x axis parameter will be bigger or smaller randomly. So this appears that the boss will move rightward or leftward randomly. The following picture is about this part.

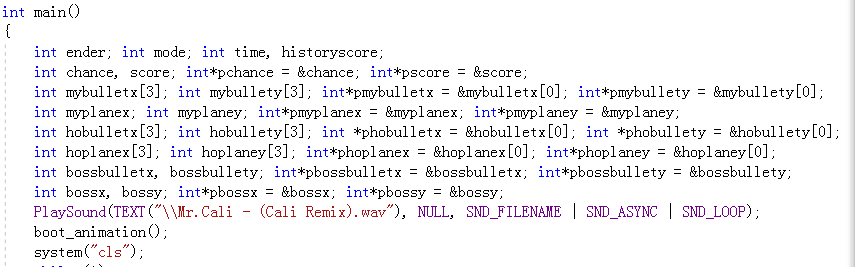
C:\Users\Administrator\AppData\Roaming\Tencent\Users\2848123049\QQ\WinTemp\RichOle\6WS]O$RTSLLU_E)U{A1%%RN.png

As for the last function, boss check, its mechanism is also the same as the former corresponding one.

12.main

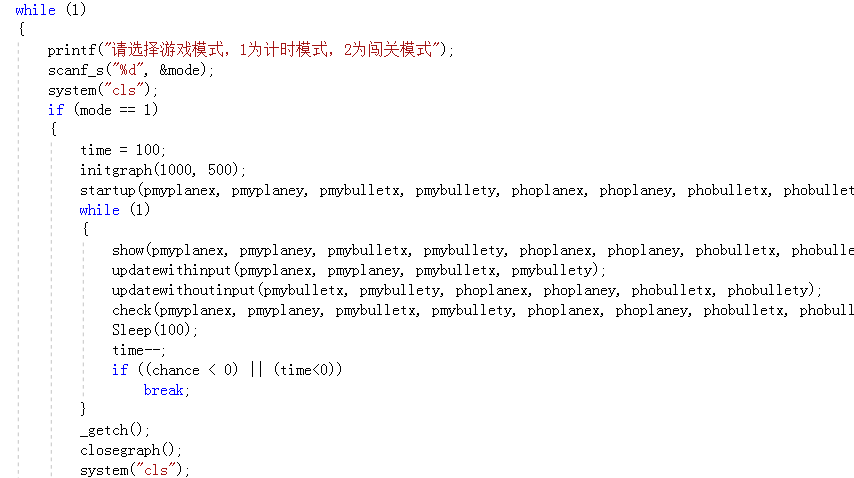
Finally, we will explain our main function.

In the first part, we declare all the parameters and their corresponding pointers. We use pointers so we will avoid using global parameters. The following picture is about this part.



Then we continue to look at the above picture. We play the background music. The music will play again and again until we close the game.

Next we play the boot animation. And then we use “system “(“cls”)” to clear the screen to prepare for the next function.



Then, we will execute the game. We look at the above picture. First we print “choose mode” to let player to choose. And we scan what player presses. Then we clean the screen again. If the player chooses the timekeeping mode, mode will equals to 1. And we will execute the mode one.

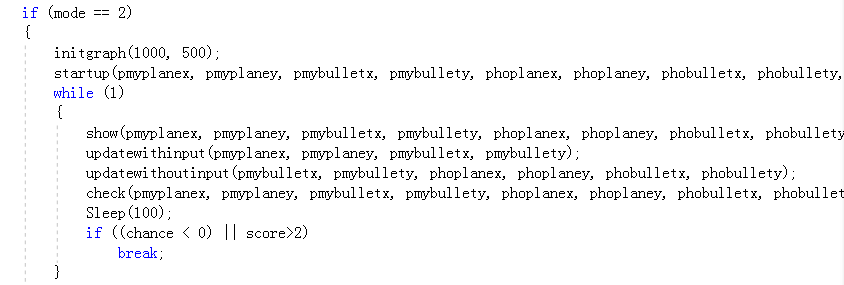
We let the parameter “time” equals to 100. Because in every loop, we let the program sleep 0.1 second, so in total this mode will last 10 second. Then we use initgraph function to open a space for easyx function to draw planes. Next we execute the startup function to initialize all the parameters. And then we execute the game loop. We will execute these functions: show, update with input, update without input, cheek, sleep and time—again and again. When chance equals to 0 or time equals 0, the loop will break. And the player will have to press one key on the keyboard and we will close the graph space then we clean the screen again.

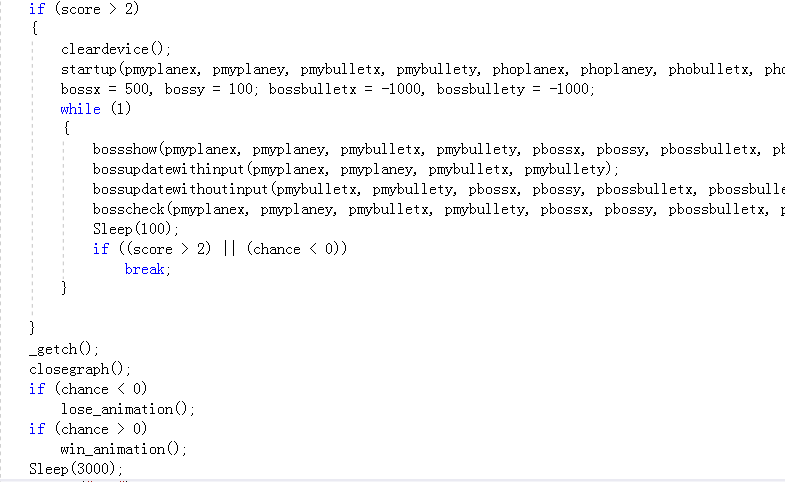
Then we go into the showing score part. We will look at the following picture.



In the picture, we can see that first we will check if there is a file saving history score. If there is not, we build a new file and let the history score equal to 0. If there is a file saving score, we read the file and let history score equal to what we read. Then we compare the new score and the history score. If the new score is bigger than history score, we let the history score equal to new score. Then, we print the new score and the history score.

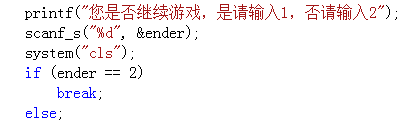
If player choose mode 2, the first half part of the loop is the same as the mode 1. But when the chance is smaller than 0 or the score is bigger than 2 we break the loop. Like the following picture.



Then, there are two conditions. If the chance is smaller than 0, we will execute lose animation function. There will be a “lose” printed on the screen. 

We will look at the above picture. If the score is bigger than 2, we will execute the boss part. In the beginning, we will execute startup function. This means that all parameters will be initialized. The boss loop’s mechanism is the same as the former loop. When the score is bigger than 2 or the chance is smaller than 0, the boss loop will break. If the chance is smaller than 0, the “lose” will be printed. If the score is bigger than 2, the “win!” will be printed.

Now we look at the following picture. At the end of the main function, we will let the player to input “1” or “2”. If player presses 2, the loop will break. If the player presses 1, the main function will be executed again. What we should pay attention to is that, we use “while (1)” at the beginning of the main function. So actually the main function is the biggest loop. In the biggest loop, there are many sub loops. So in the following picture, we do not write anything after else. Because if we write nothing, we will go back to the begging of the main function and execute it again.



And by now, the implementation has been finished.

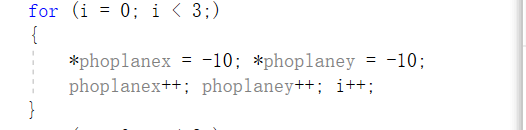
## Testing and Debugging

Test

Our test of the program can be divided into three parts. The first part of the test is the test of the “for” loops in every function, this kind of test can make sure that in each part of the function, our program changes the parameters correctly. And then is the test of each function. if you have read the report above, you have already know that the game loop involve four important functions: show, updatewithinput, updatewithoutinput and check. We pass the pointers to each function and then check if the function has changed the parameter according to the pointers. Finally, we run the whole program to check whether it can work successfully.

the test of the small loops

Our plane has three bullets. There are three hostile planes and each of them has one bullet. So most of times we should change more than one parameters that have the same function in one step. And we always use the “for loop” to do this kind of work .For example, we have these codes below:



This loop is used to give the original value of the location parameters about the hostile planes. Since we use the pointers, the meaningful change of the parameters happens at the location where the pointer points to. And if we check how the \*phoplanex changes, you can see the hoplanex[0] changes one by one just like this:

|  |  |  |
| --- | --- | --- |
| The value of the parameter  before the ‘for loop’ | The value of the parameter  After the ‘for loop’(expected) | The value of the parameter  After the ‘for loop’(real) |
| The parameter has no meaningful value | -10 | -10 |
| The parameter has no meaningful value | -10 | -10 |

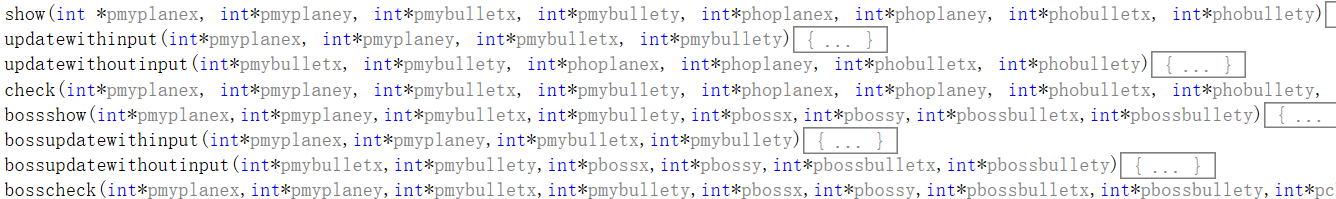
So you can see, if we check the value of the phoplanex[0] before this for loop, the computer will return some numbers randomly . But if we check the phoplanex[0] after the for loop, the computer will return -10.

This kind of test is very easy because we just need to check what happens to one parameter of a group of parameters. This is because we change the parameters in the same way, and if one of them changes correctly, the rest of them will also changes in the same and correct way.

The test of the functions

The functions that has no input

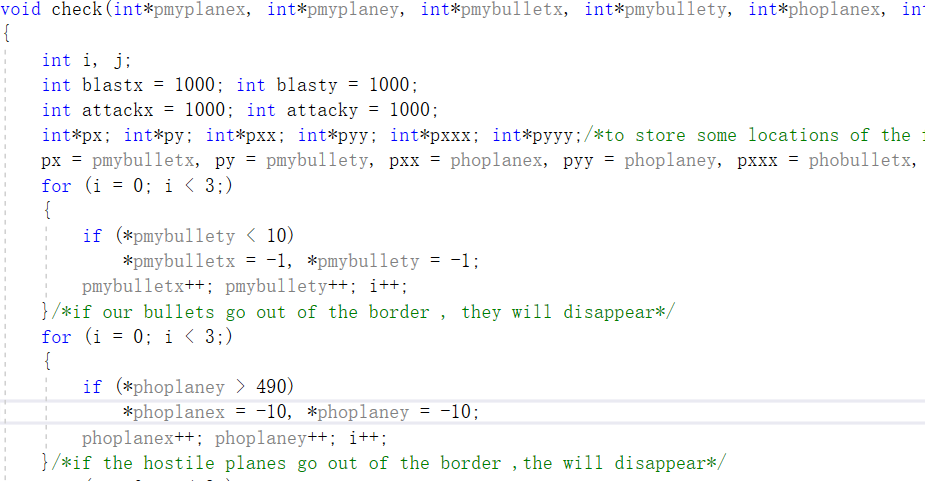
This part of the tests is the most difficult part of the test . All the parameters in these functions are not input by the players. And in fact, all the meaningful parameters are defined in the main function. But in each functions such as ‘show’ and ‘check’, we need to use them and even change them. So we use the pointers and all the parameters that we passes to the ‘show’ and ‘check’ are pointers. And the number of the pointers we pass is very big:



So we check the parameters in one function only one or only one group at a time. For example,we test how the mybulletx changes in the function ‘updatewithoutinput’:

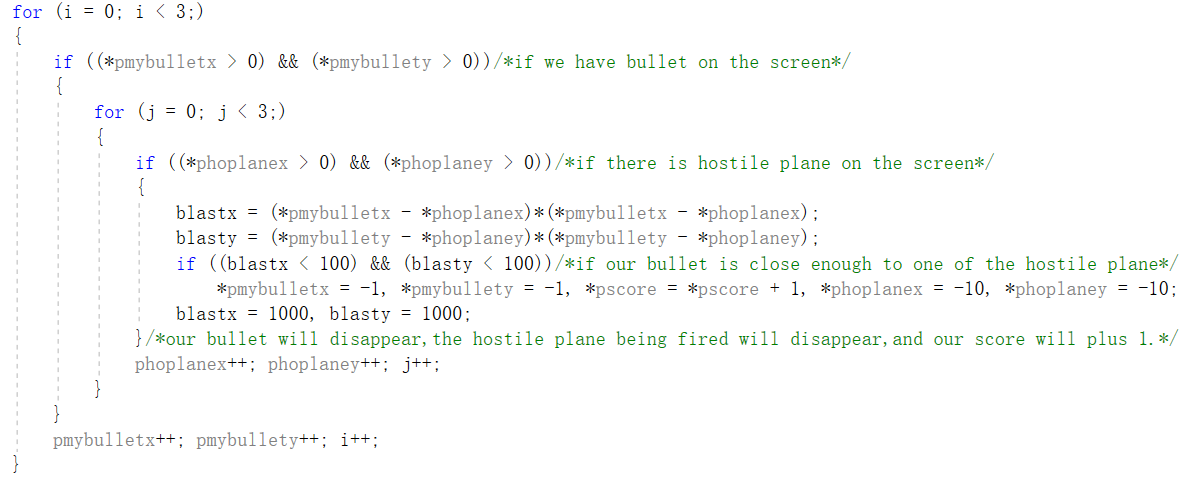
|  |  |  |
| --- | --- | --- |
| The value of mybulletx before ‘updatewithoutinput’ is executed | The value of mybulletx after ‘updatewithoutinput’ is executed(expected) | The value of mybulletx after ‘updatewithoutinput’ is executed(real) |
| -1 | -1 | -1 |
| 366 | 366 | 366 |

This is a simple example because the parameter ‘mybulletx’ is smaller than 0 if the player didn’t fire. That means the ‘show’ function will not draw it. And if the player fires, since the bullet will goes towards the top of the screen directly, the x coordinate of the bullet will not change at all. So if the value of this parameter didn’t change, the related part of the function works well.

But if the parameter should be changed in the function, we will check it more carefully. For example:

This is part of the check function , and you can see two ‘for loop’ in this picture .And according to the annotation , the first ‘for loop ’ is used to check if the bullets of the player have gone out of the border . And if the answer is yes, then the function will change the coordinate parameters of the bullet to -1. And we all know that if the coordinate parameter is smaller than 0, the ‘show’ function will not draw it. So in this way, the ‘check’ function makes the bullets disappear. And similarly, the coordinate parameters of the hostile planes will also be changed to -10 if the hostile plane goes out of the border.

However, in the whole function, there are other parts that the ‘mybullety’ and ‘hoplaney’ is involved. The codes below are also from the ‘check’ function:



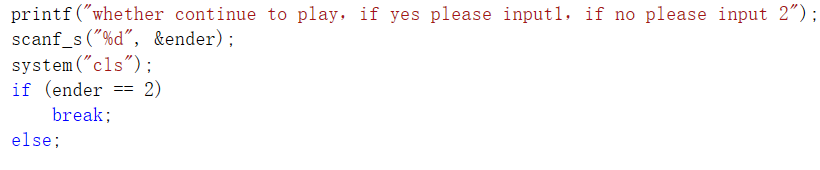
This part is checking whether our bullets have hit the hostile planes. And if the answer is yes , the function will make both the hostile plane and our bullet disappear .

So if we test how the ‘myplaney’ and ‘hoplaney’ change in the ‘check’ function:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| mybullety[0] before check is executed | mybullety[0] after check is executed(expected) | mybullety[0] after check is executed(real) | hoplaney[0] before check is executed | hoplaney[0] after check is executed(expected) | hoplaney[0] after check is executed  (real) |
| -1 | -1 | -1 | -10 | -10 | -10 |
| 220 | 210 | 210 | -10 | -10 | -10 |
| -1 | -1 | -1 | 50 | 60 | 60 |
| 270 | 260 | 260 | 150 | 160 | 160 |
| 180 | -1 | -1 | 185 | -10 | -10 |

The function that has input

In the main function, some parameters are input by the player. For example , the parameter ‘ender ’ :

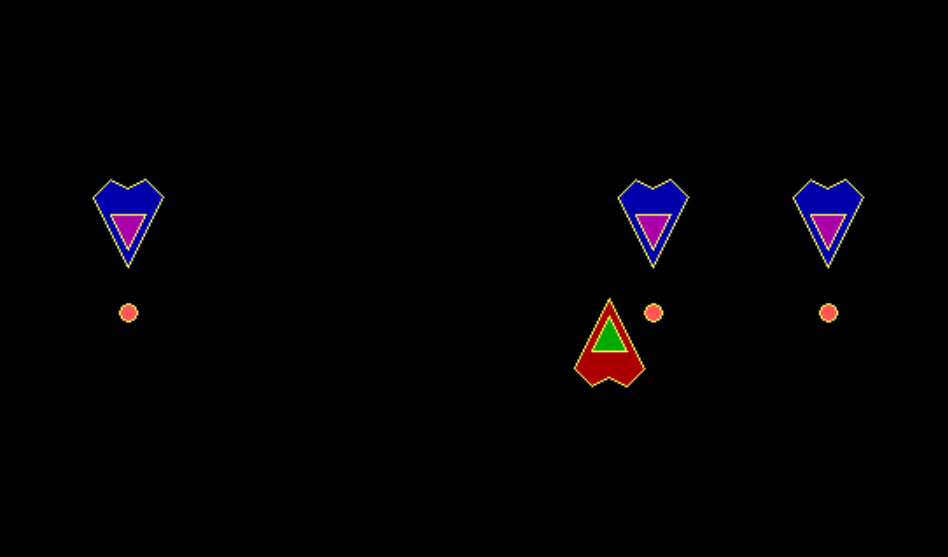


In the main function, the original value of the ‘ender’ is 0. And when we test this part:

|  |  |  |
| --- | --- | --- |
| The input of ‘ender’ | What we expected to happen | What happened in fact |
| 1 | The game continues | The game continues |
| 2 | The program ends | The program ends |
| 3(illegal input) | The game continues | The game continues |

Although the input ‘3’ is illegal, the program happens to work well because the program just checks if it should break from the loop. It doesn’t care about whether it should continue.

The test of the whole program

Finally, we put all the codes together and tests if the program works well. And luckily there is on error.

## Result&Conclusion

The aim of our project which is running a game has already achieved .In this game, both our plane and hostile plane emerge on the screen, and we can control the movement of airplanes and the firing of bullets. What’s more, the hostile planes can also move to us and fire the bullets. The design of two mode is also achieve, and in the timekeeping mode we will defeat more enemy aircraft in limited time and challenge other people’s score, and in the boss mode when we defeat boss we will win. We also solve the problem in some other games that the screen flashes. Different modes make the game more interesting and more playable. Adding the right music makes the game more immersive. Of course our game still has some flaws. First of all, when enemy aircraft appear, they might overlap. We think we can solve this problem by changing the random function. What’s more, our game is a little monotonous. To solve this problem, we decided to design much more selections with different difficulties and add some props for example we plan to add a kind of prop and get it so that we can fire multiple bullets at the same time. Our interface is a little rough. In next step we will make the interface more beautiful.

After the programing, we have learnt a lot. We not only learn the knowledge from class, but also learn more knowledge from our classmates, professor Yibo and from the internet. When we met problems, we learn to find the answers or the solutions by ourselves. I think this is one of what we have learnt in the C programing class. The next thing we have learnt is that we learn to collaborate better with our classmates. In the programing process, we write different codes. And we learn how to divide the works and how to link them. The final thing we learned is how to express us properly, how to give a presentation and to grab the audience’s attention. When we give the presentation, we should meet what professor demand. Now the programing is done. But I think that we will often use the C programing and we will never stop learning the C programing in the future. Finally thanks to professor Hao, professor Yibo and Professor AT.