# C Project Report: Bean eating

## Abstract

The Pac-Man game is a very classical video game from Japan, since 1980s. For those video gamers from all over the world, this game is one of their childhood memories. When we listening the familiar music, we trend towards avoid the ghosts and eat more beans.

Our group, since half of us are the video gamer, we decide to make this game again, so that we can get the previous and precious feelings again.

In our game programming, we met some central problems. The very first one is, as we are building our video game, the graphical surface is the important thing. And then we came across the problems about control the movement of the ghosts and the Pac-Man. Of course there are more problems during the game forming, these two are the most essential ones.

In our new Pac-Man game, we have three maps for players to try. We also tell you how many scores that you totally get. As a senior video gamer, I’d like we can have the classical games been a new one, so that we can double our fun.

## Introduction & Problem Statement

In our childhood, video games maybe one of the best friends of us. Pac-man is a pretty classical one and it just like an old friend of us, it accompany most of our childhood with the classical background music and take an important position in us video gamers memory. It’s an arcade game first developed by Namco and released in Japan in May 1980.

As a result, after we have learned Language C, we decided to try our best to repeat this game with Language C and add some interesting factors into it.

To a game, I think the most significant factors are the interactivity and gamer experience. The goal of this game is to accumulate points by eating all the Dots in the maze and the four ghosts roam the maze, trying to kill Pac-Man.

Above all, we need to design a graphical surface. This is really important. In fact, for the traditional C language, it’s really weak in graphical showing. Most of this kind of game is built by Python or Java, seldom in C. So that it’s really a challenge for us because there’s not so much proper code for us to consult. After preparing and referring to some datum, we decided to use EasyX in C++ to release the graphical surface.

What’s more, the movement control is also the big problem. On account of there is little source code, we need to think of all the arithmetic about the moving judgement by ourselves. The final aim of our project is to complete the whole game and pass the happiness to these video gamers

## 2. Analysis

We all know that the goal of this game is to control the Pac-man eat all the Pac-dots in the maze and elude the ghosts smartly at the same time. The essence of this game is just between the moment of eating and eluding.

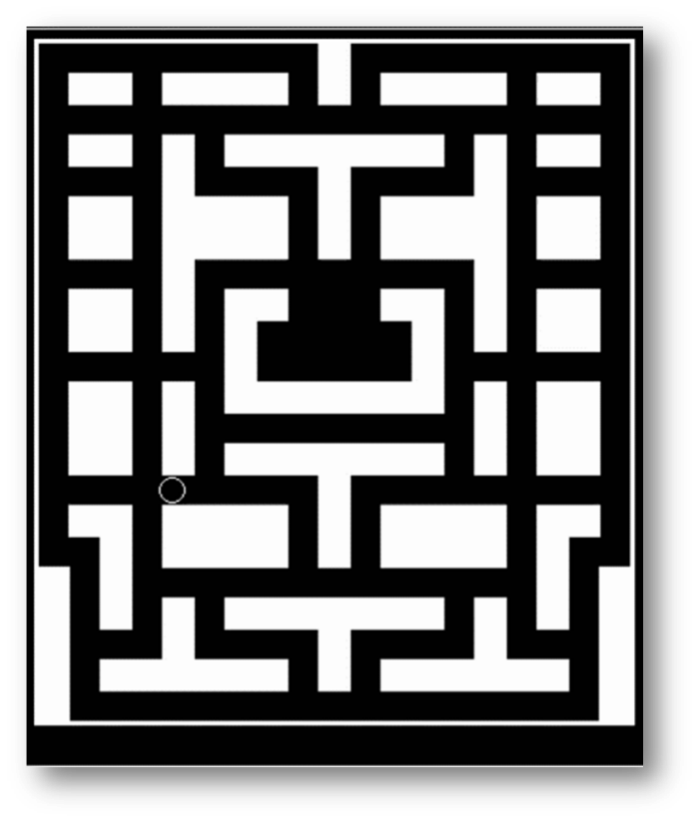
The operation of this game is extremely easy although it still needs a good control if you want to pass the game. The Pac-man moves in a certain speed and you just need to press W A S D to change the moving direction. The Pac-man can’t cross the wall and the ghost. If the ghosts catch the Pac-man, the Pac-man’s lives will minus one and the game will terminate when the number of lives comes to zero. Only when the Pac-man eaten all the Pac-dots in the maze, it can come into the next level and there are three levels in this game.

Compared to the original game, the game we design is not as difficult as the original one. The original will increase the number of the ghosts in the next level when you have finished one level; it highly improves the possibility of caught by the ghosts.

Now that we have decided the main direction, we need to try to understand the problems on hand and supposed to be solved.

* Build a game interface and draw the maze in the map

As a game, of course it needs a graphical surface to have interactivity with the player. What’s more, the map of the game must be showed completely in the map. So that we use EasyX to initialize a graphical surface and begin to design a symbol-like game with some simple shapes like circle and rectangle. We control everything on the interface by their coordinates. The interface is 480 Dots×640 Dots.



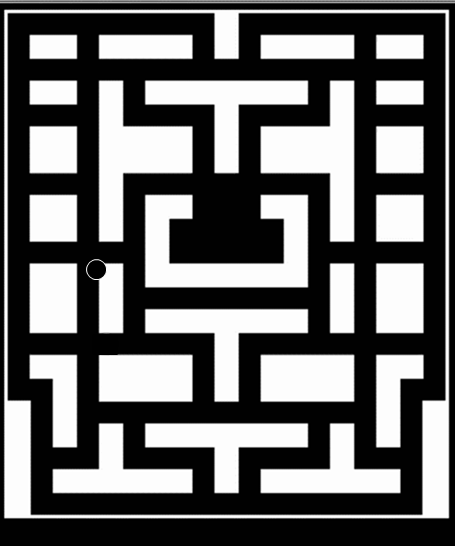
* The movement of the ghost and the Pac-man

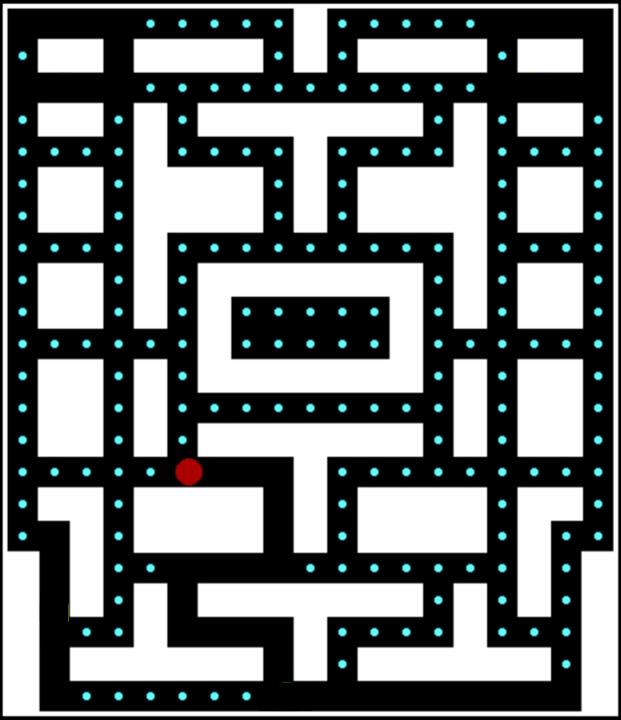
As the basic part of this game, the movement of the Pac-man and the ghosts can be said the most important. How to make these characters move together and fluently is the most complex part of our program because the program is single thread.

What’s more, the movement of the ghost should be random but it cannot produce the random number frequently because it may turn around at one place all the time. As a result, we decide to make the judgement at the crossing. Then how to judge a crossing? That’s quite a problem to us.

In addition, the movement of the Pac-man is controlled by our input commands. We need to press W A S D to change its direction. But we can’t make the Pac-man hit on the wall or sweep the wall directly. We need it follows the player’s commands then turn left or right at the center of the crossing.

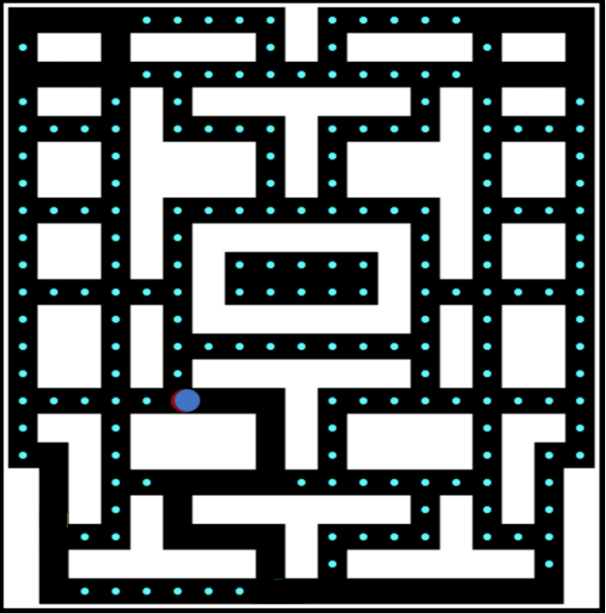
* The ghosts will sweep the wall and the Pac-dots

Above all, if we want to know why there exists this problem, we need to understand the theory of the movement of the ghosts. It moves by wiping off the original image and drawing a new one just next to it. But if it can’t judge whether the four directions are a wall or not, the movement of the ghost will cause the wall to be swept. The Pac-dots be swept is the same kind of problem like that one.



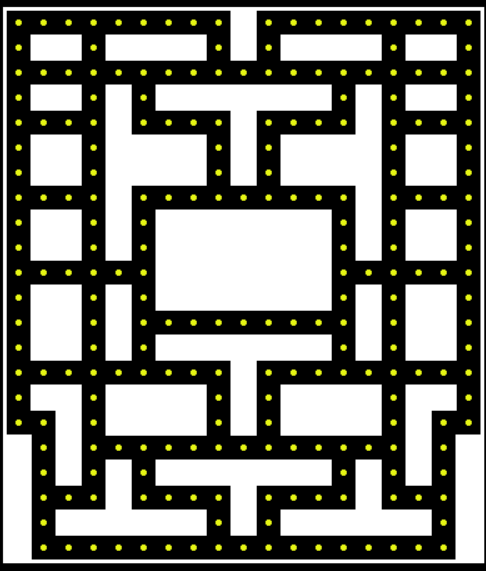
* The judgement for death and game over

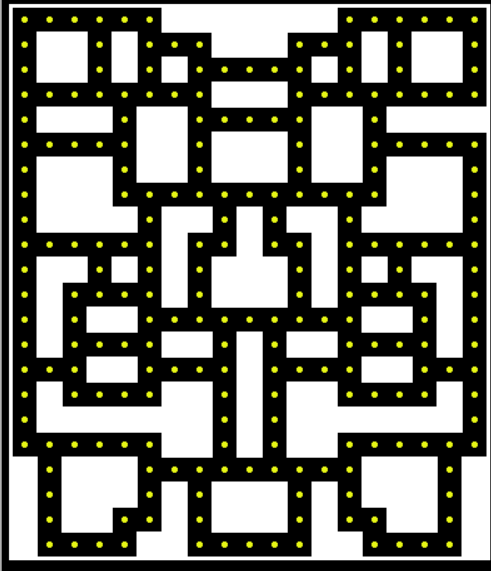
The death in this game likes all the games in this kind, if you were caught by the ghosts, you dead and when you have no lives, game is over.

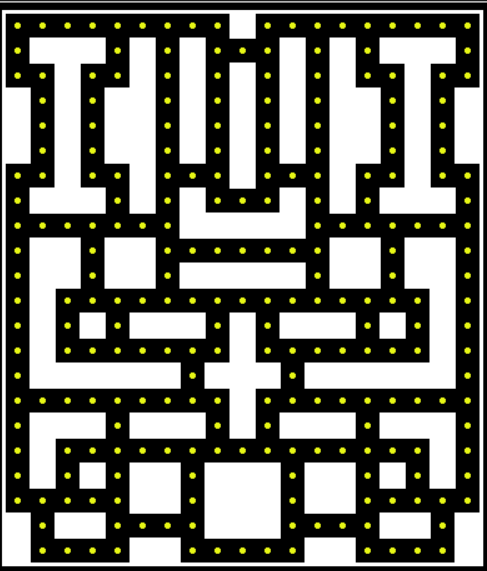


* The switch between the levels and the winning judgement

For this game, it’s too simple to have only one level for us players so we designed three maps as three levels and we need to design arithmetic for the switch. When you pass these three levels, you win the game.

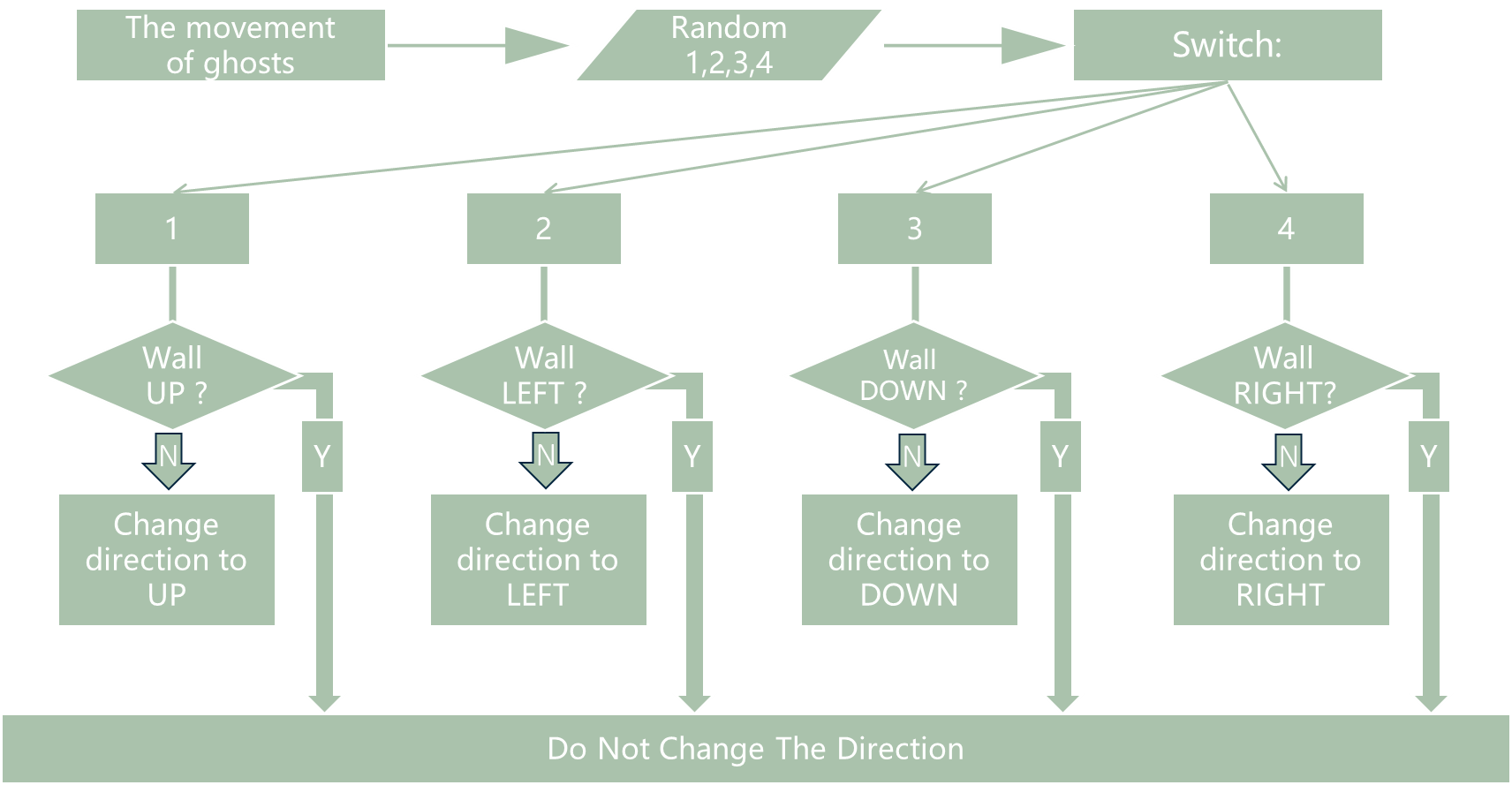




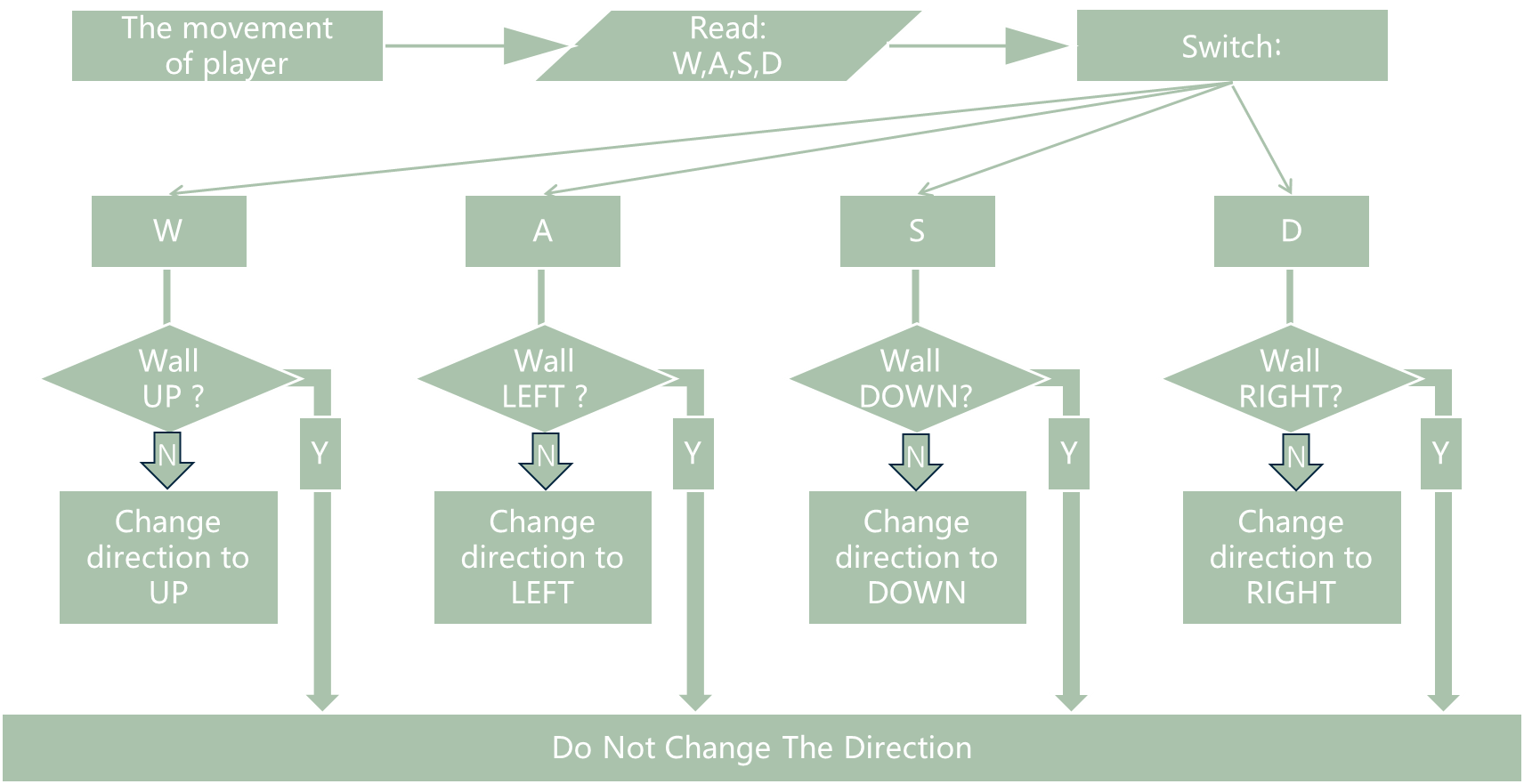


## 3.Design

The movement of the ghosts and the Pac-man



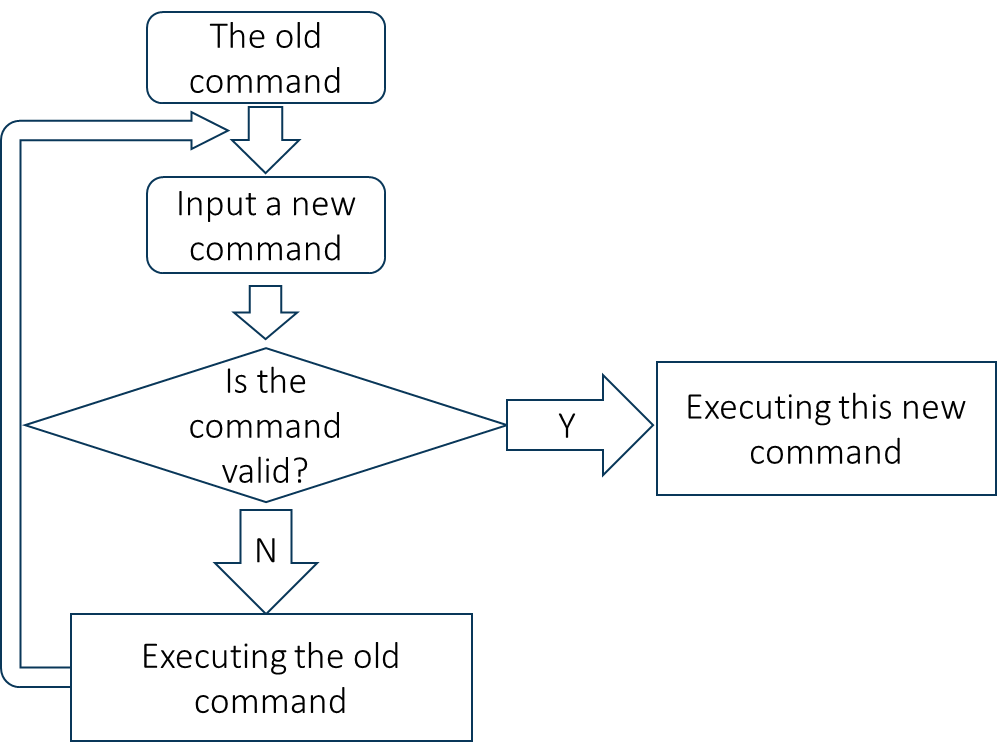
As for the movement, we use many functions called “if- function”. If the pot of the heading direction is colored with white, the direction is unavailable. And if one of the X or Y coordinate of the ghost or Pac-man can’t be exact division by 20, we can say that his position is not at the center of any lattices, then surely, he can’t change the direction. More interestingly, wo should know if ghost is at the cross, if not, do not change the direction, so that we can make the game more playable, otherwise, the ghost maybe fall into a “trap” of even a normal road.



After we have determined the direction, we draw a black circle over the original place to cover the image, and then draw a new image at a distance from a pixel to achieve ghost and Pac-man’s movement.

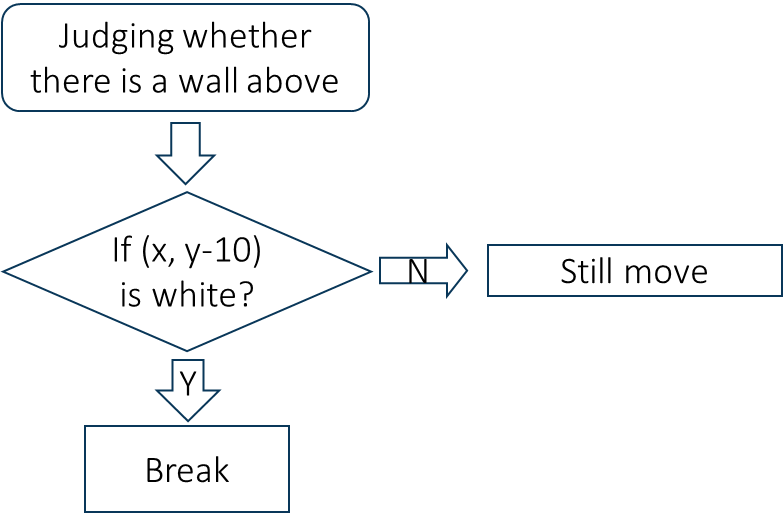
* The optimization for the movement of the Pac-man

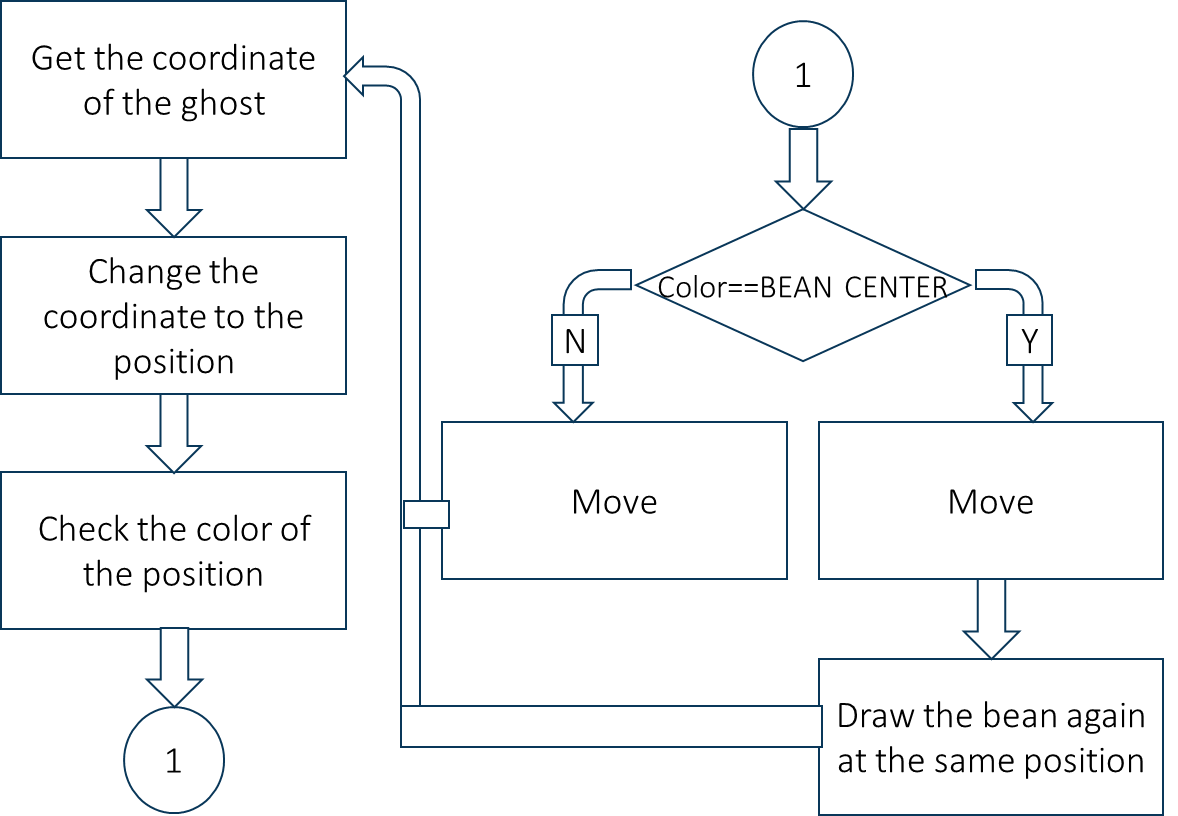
In the beginning, our design was to press W A S D to control the direction of movement. If there is no new command, it will move in the previous direction. But we have a moving judgment that if the Pac-man’s coordinate is not an integer multiple of 20, the Pac-man will not move in the command direction, but the previous command has been overwritten, so the Pac-man will be stationary. So, how do we solve this question? When we make the movement command judgment, if it’s invalid, we will ignore it and save it then we can do it again. It’s a loop and the command will be executed until it’s valid.



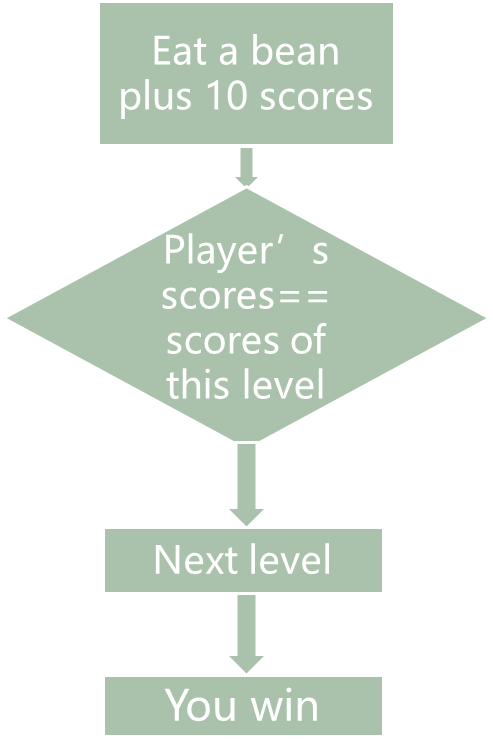
* The ghosts will sweep the wall and the Pac-dots

The ghost moves by wiping off the original image and drawing a new one next to it. If it can’t judge whether he four directions are a wall or not, the movement of the ghost will cause the wall to be swept. Thus we design a judgement to make sure whether it’s the wall around the ghosts. We use a function in EasyX called **getpixel** to judge the color around the ghosts. As I said that we can control all the things in this game interface by coordinate. This judgement we just use the coordinate to check the wall around the ghosts conveniently.

 This problem is quite similar. At first we decided to draw a black ghost in the previous position, but we do not draw the Pac-dots. So we will draw the Pac-dots again at the same position after the judgement about the Bean-Color.

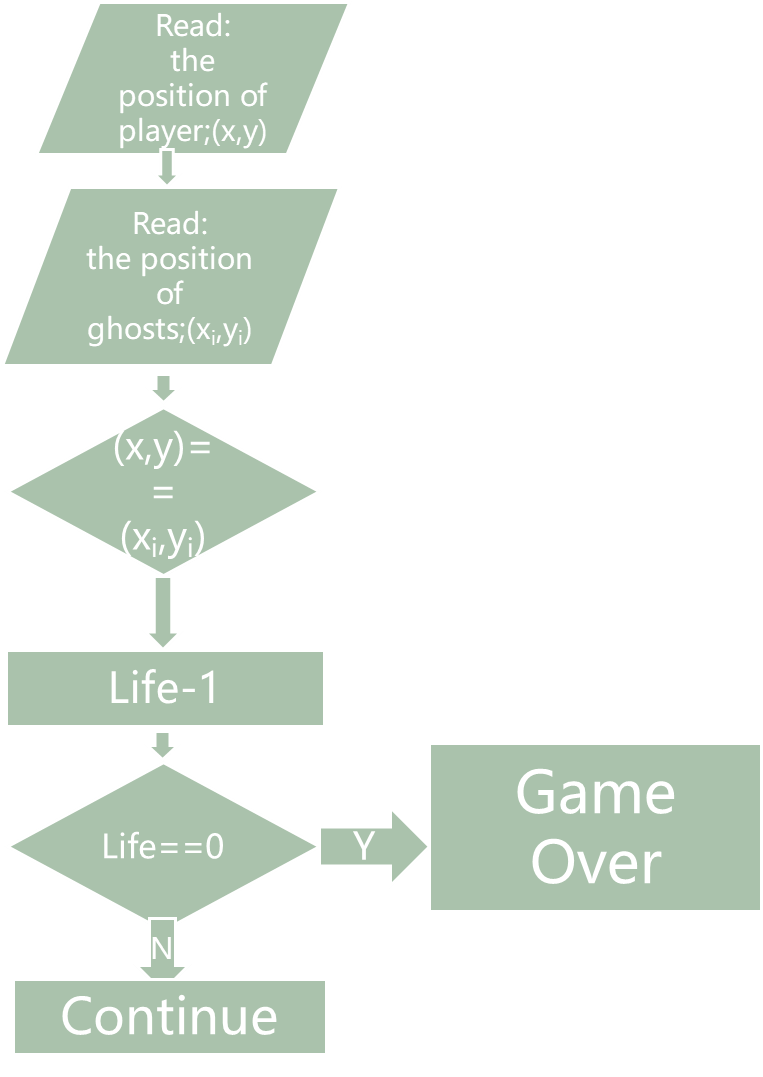


* The switch between the levels and the winning judgement

We design arithmetic to accumulate scores you get during playing the game as the winning judgement. When the Pac-man eats one Pac-dot, present score will plus ten. At first, we will calculate the number of Pac-dots and record the total score. If your present score reach the total score, then comes to the next level. The final winning standard is completing all the levels and left at least one life.

* The judgement for death and game over

As we all know that if the Pac-man caught by the ghosts, its lives will minus one. As a result, we design a judgement that if the coordinate of the Pac-Man is the same as one of the ghosts, we will refresh the position back to the initialization settings, and then check the life number, if it is less than 1, you will fail the game, and if not, you can continue and 1 will subtract form the lift numbers. If the coordinate of the Pac-Man is different from any of the ghosts, then we will check if you score is enough to go to the next level. If it isn’t enough, the loop will come back to the player input.



**4.Implementation**

1. Interface (Map Drawing)

In terms of how to implement the mapping, we searched the Internet for a lot of information and finally decided to draw it with EasyX. In order to be able to initialize the graphical interface, the following code is essential.

int driver, mode;

driver = DETECT;

mode = 0;

initgraph(&driver, &mode, ""); // Initialize the graphical interface

Then let me show a small part of the code of our map.

line(1, 1, MAX\_WIDTH - 1, 1);

line(1, 1, 1, MAX\_LENGTH - 1);

line(1, MAX\_LENGTH - 1, MAX\_WIDTH - 1, MAX\_LENGTH - 1);

line(MAX\_WIDTH - 1, 1, MAX\_WIDTH - 1, MAX\_LENGTH - 1);

line(10, 10, WIDTH + 10, 10);

line(10, 10, 10, LENGTH + 10);

line(10, LENGTH + 10, WIDTH + 10, LENGTH + 10);

line(WIDTH + 10, 10, WIDTH + 10, LENGTH + 10);

rectangle(9, 9, 391, 451);

rectangle(8, 8, 392, 452);

rectangle(30, 30, 70, 50);

bar(30, 30, 70, 50);

rectangle(90, 30, 170, 50);

bar(90, 30, 170, 50);

rectangle(190, 10, 210, 50);

bar(190, 10, 210, 50);

rectangle(230, 30, 310, 50);

bar(230, 30, 310, 50);

rectangle(330, 30, 370, 50);

bar(330, 30, 370, 50);

rectangle(30, 70, 70, 90);

bar(30, 70, 70, 90);

rectangle(90, 70, 110, 210);

bar(90, 70, 110, 210);

rectangle(130, 70, 270, 90);

bar(130, 70, 270, 90);

rectangle(290, 70, 310, 210);

bar(290, 70, 310, 210);

rectangle(330, 70, 370, 90);

bar(330, 70, 370, 90);

Some of the features mentioned next have already been mentioned in the analysis section, and I will explain in detail how it was achieved.

2) Pac-Man and Ghosts Judging Walls on the Move

If the ghosts and Pac-Man cannot accurately judge the walls, then there will be a big problem in the process of their movement - the appearance of eating the walls. The ghost moves by wiping off the original image and drawing a new one. If it can’t judge whether the four directions are a wall or not, the movement of the ghost will cause the wall to be eaten. So how do we accurately determine where walls are?

Let’s see our code for judging the position of walls.

int if\_wall(int x, int y) // To determine if it is a wall

{

if (getpixel(x, y) == WHITE || ((x + 10) == 400) || ((x - 10) == 0) || ((y + 10) == 460) || ((y - 10) == 0)) // If it is white or beyond the border

return 1; // Return true

return 0;

}

int if\_wall\_2(int x, int y) // Judging the wall position

{

int w = 0, a = 0, s = 0, d = 0;

if (getpixel(x + 10, y) == WHITE || ((x + 10) == 400)) // There is a wall on the right

d++;

if (getpixel(x, y + 10) == WHITE || ((y + 10) == 460)) // There is a wall below

s++;

if (getpixel(x - 10, y) == WHITE || ((x - 10) == 0)) // There is a wall on the left

a++;

if (getpixel(x, y - 10) == WHITE || ((y - 10) == 0)) // There is a wall on the top

w++;

if ((a == 1 && d == 1) || (w == 1 && s == 1))

return 1;

return 0;

We determine the position of the wall by judging the RGB color value of the pixel at the specified coordinate point and judging whether the position of the ghost and the pac-man side sheet exceeds the boundary. Because the wall is white, you can judge the color to see if it is a wall. However, there is also a situation where the ghost and Pac-Man reach the border of the map. If this situation is not added to the judgment, ghosts and Pac-Man will also default at the boundary, which will cause ghosts and Pac-Man to reach the map outside.

The following code is used to determine whether ghosts and Pac-Man are horizontal and vertical grid points and integer grids to achieve the movement of ghosts and Pac-Man. This code, combined with the above judging the position of wall’s code, enables the ghosts and Pac-Man to run against the wall during exercise and change direction or turn.

int if\_run(int x, int y)

{

if (x % 20 != 0)

return 1; //return 1，can’t go up and down

else if (y % 20 != 0)

return 2; //return 2，can’t go right and left

else

return 0; //return 0，Any direction

}

int if\_run\_2(int x, int y)

{

if (x % 20 != 0 || y % 20 != 0)

return 0;

return 1;

}

3) How ghosts do not eat beans in action

The ghost moves by wiping off the original image and drawing a new one. At first we decided to draw a black ghost in the previous position, but we do not draw the bean. So we have adopted a new method - complementary beans. By the success of judging the wall, it is much simpler to implement this function.

The following flow chart is our thinking and algorith.

And this is our code for the implement.

setcolor(color);

setfillcolor(color); // Set the graphic color

wall\_num = if\_wall\_2(\*x, \*y); //Determine whether it is a junction

n = if\_run\_2(\*x, \*y); // Determine if it is an integer grid

if (!wall\_num && n) // Integer grid point and change the random number at the intersection

{

srand((unsigned)time(NULL)+ random); // Set random seed

\*dir = (rand() % 4 + 1); // Generate random numbers

}

bean = if\_bean(\*x, \*y, BEANCOLOR); // Bean determination

switch (\*dir) // Direction determination

{

case 1:

{

wall = if\_wall(\*x, \*y - 10); // Judging whether there is a wall above

if (!wall)

go\_up\_g(x, y, color, SPEED\_V, bean);

break;

The next cases 2, 3, and 4 are walls that judge different directions, and they will not be listed one by one.

4) The death and the end of the game

We stipulated that the Pac-Man's life is 3, and each time it is eaten by a ghost, it reduces the number of lives. When the number of hits is 0, the game is over and the end page is displayed.

This is our code for the death of pac-man.

if ((\*player\_x1 == \*ghost\_x1 && \*player\_y1 == \*ghost\_y1) || (\*player\_x1 == \*ghost\_x2 && \*player\_y1 == \*ghost\_y2) || (\*player\_x1 == \*ghost\_x3 && \*player\_y1 == \*ghost\_y3) || (\*player\_x1 == \*ghost\_x4 && \*player\_y1 == \*ghost\_y4))

{

(\*player\_Life)--;

setcolor(BLACK);

setfillcolor(BLACK);

fillrectangle(\*player\_x1 - 8, \*player\_y1 - 8, \*player\_x1 + 8, \*player\_y1 + 8);

\*player\_x1 = 180;

\*player\_y1 = 440;

setcolor(BLACK);

setfillcolor(BLACK);

5) Switch between level and winning judgment

We set aside 10 points for each bean to be eaten. When the Pacman eats all the beans and reaches all points in this pass, it will switch to the next level. We used the switch function to switch the level.

int a, b, i;

COLORREF color = 0xFACB88;

setfillcolor(BEANCOLOR);

setcolor(BEANCOLOR);

for (a = 20; a <= 380; a += 20)

for (b = 20; b <= 440; b += 20)

{

if (getpixel(a, b) == WHITE)

continue;

fillcircle(a, b, 2);

putpixel(a, b, CENTERCOLOR);

m1 += 10;

}

m1 -= 10;

outtextxy(440, 160, "level 1");

outtextxy(440, 220, " Current score：");

outtextxy(440, 280, " Remaining number of lives：");

return m1;

for (i = 1; i < 4; i++)

{

switch (i)

{

case 1:

totalscore = map\_1(totalscore);

break;

case 2:

totalscore = map\_2(totalscore);

fresh(&ghost\_1.x, &ghost\_1.y, 20, 20, CYAN, 1);

fresh(&ghost\_2.x, &ghost\_2.y, 360, 440, GREEN, 1);

fresh(&ghost\_3.x, &ghost\_3.y, 380, 20, BLUE, 1);

fresh(&ghost\_4.x, &ghost\_4.y, 40, 440, YELLOW, 1);

fresh(&player1.x, &player1.y, 180, 440, RED, 0);

break;

case 3:

totalscore = map\_3(totalscore);

fresh(&ghost\_1.x, &ghost\_1.y, 20, 20, CYAN, 1);

fresh(&ghost\_2.x, &ghost\_2.y, 360, 440, GREEN, 1);

fresh(&ghost\_3.x, &ghost\_3.y, 380, 20, BLUE, 1);

fresh(&ghost\_4.x, &ghost\_4.y, 40, 440, YELLOW, 1);

fresh(&player1.x, &player1.y, 180, 440, RED, 0);

break;

default:

break;

}

while (true)

{

if ( \_kbhit())

d = \_getch();

for (pl = 0; pl < 2; pl++)

player\_control(&player1.x, &player1.y, RED, &player1.direct,&d);

ghost\_control(&ghost\_1.x, &ghost\_1.y, CYAN, &ghost\_1.direct, 1);

ghost\_control(&ghost\_2.x, &ghost\_2.y, GREEN, &ghost\_2.direct, 2);

ghost\_control(&ghost\_3.x, &ghost\_3.y, BLUE, &ghost\_3.direct, 3);

ghost\_control(&ghost\_4.x, &ghost\_4.y, YELLOW, &ghost\_4.direct, 4);

j=initialize\_1(&player1.x, &player1.y, &ghost\_1.x, &ghost\_1.y, &ghost\_2.x, &ghost\_2.y, &ghost\_3.x, &ghost\_3.y, &ghost\_4.x, &ghost\_4.y, &player1.life);

if (j)

{

closegraph();

break;

}

**5.Testing&Debuging**

1. Two ghosts in the second level didn’t move

|  |  |  |
| --- | --- | --- |
| Situation | Expected Result | Real Result |
| Debugging switching level is normal | The game can be normal cut off and the interface shows no problem, Pac-Man and ghosts can move normally | The second level has two ghosts that are not moving, but it still has the ability to eat Pac-Man. |

This problem also plagued us for a long time. Later it was discovered that there was a change in the second-level map, and whether the position of the ghost-initialized position was an integer lattice point had a problem. The judgment failed and the ghost could not be moved.

2) Go into the next level inaccurately

|  |  |  |
| --- | --- | --- |
| Situation | Expected Result | Real Result |
| Debugging switching level is normal | You can eat all the beans and switch to the next level | It is often the case that not all beans are eaten, but it switched to the next level by the Pac-Man, or there is no cut after all the beans have been eaten. |

Regarding this issue, due to the problems of scores and how to calculate scores if the bean is eaten half by the Pacman, we have made several improvements to the conditions of the customs, and after many attempts and adjustments, we finally succeeded.

1. Death issue

|  |  |  |
| --- | --- | --- |
| Situation | Expected Result | Real Result |
| Test whether Pac-Man can normally be eaten by ghosts | There is no problem, ghosts can eat Beanie | If Pac-Man is extremely fast, it can pass directly through the ghosts and the number of lives will not decrease. |

We have tried different ideas and methods to solve this problem, such as adjusting speed and the like. Unfortunately, we have not yet been able to solve it, it may be the speed of the program processing tasks.

4）Pac-Man's Unlimited Life

|  |  |  |
| --- | --- | --- |
| Situation | Expected Result | Real Result |
| Jump out of the game after testing for normal death | After three deaths, the game can be successfully ended | Pac-Man has unlimited lives and will not jump out of the loop to end the game |

We have finally found the error through debugging again and again. In the previous initialization function, the variable that specifies the life of pacman is inaccurately written, resulting in the program being unable to accurately identify and causing subsequent errors

**6.Result & Conclusion**

Over a period of more than three months, all of our members have spent a lot on projects. We made progress together and learned together. From the very beginning, we did not even know which tools to choose to draw maps, at the end of the game's mature operation, everyone contributed. We have successfully implemented the mapping of maps, the movement of ghosts and people, the optimization of player control programs, the realization of cut-off scoring, the judgment of death, and the ending of games. From the very beginning, it was difficult to test the game and it was very hard until the later tests. We have been able to make progress in such a short time because people often discuss and exchange ideas together. The ideas are continuously updated and matured during the exchanges, which makes our games run successfully. Now we have designed this Pac-Man game that is suitable for people of all ages. It has a beautiful interface, free movement of characters, and accumulation of the remaining number of life points and scores. It will remind you to enter the next level at each pass. The difficulty of this game is moderate. It is a good idea to play this game on your spare time.

In a word, we completed this game with our common efforts and achieved our goal at that time.