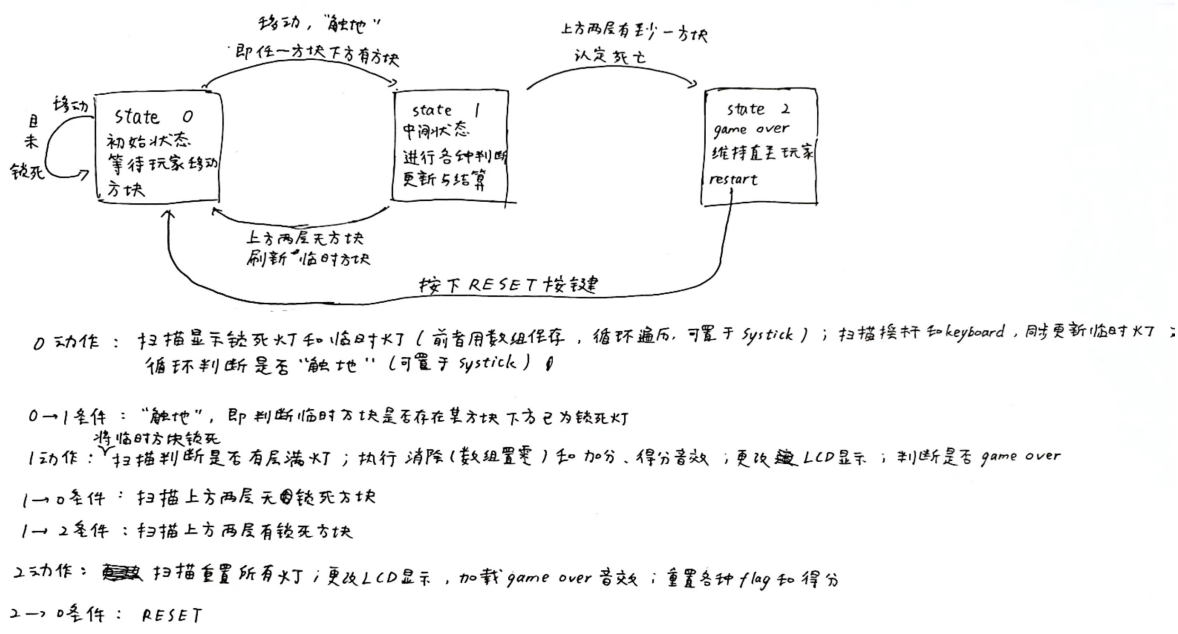
3D Tetris 软件实现

状态机：



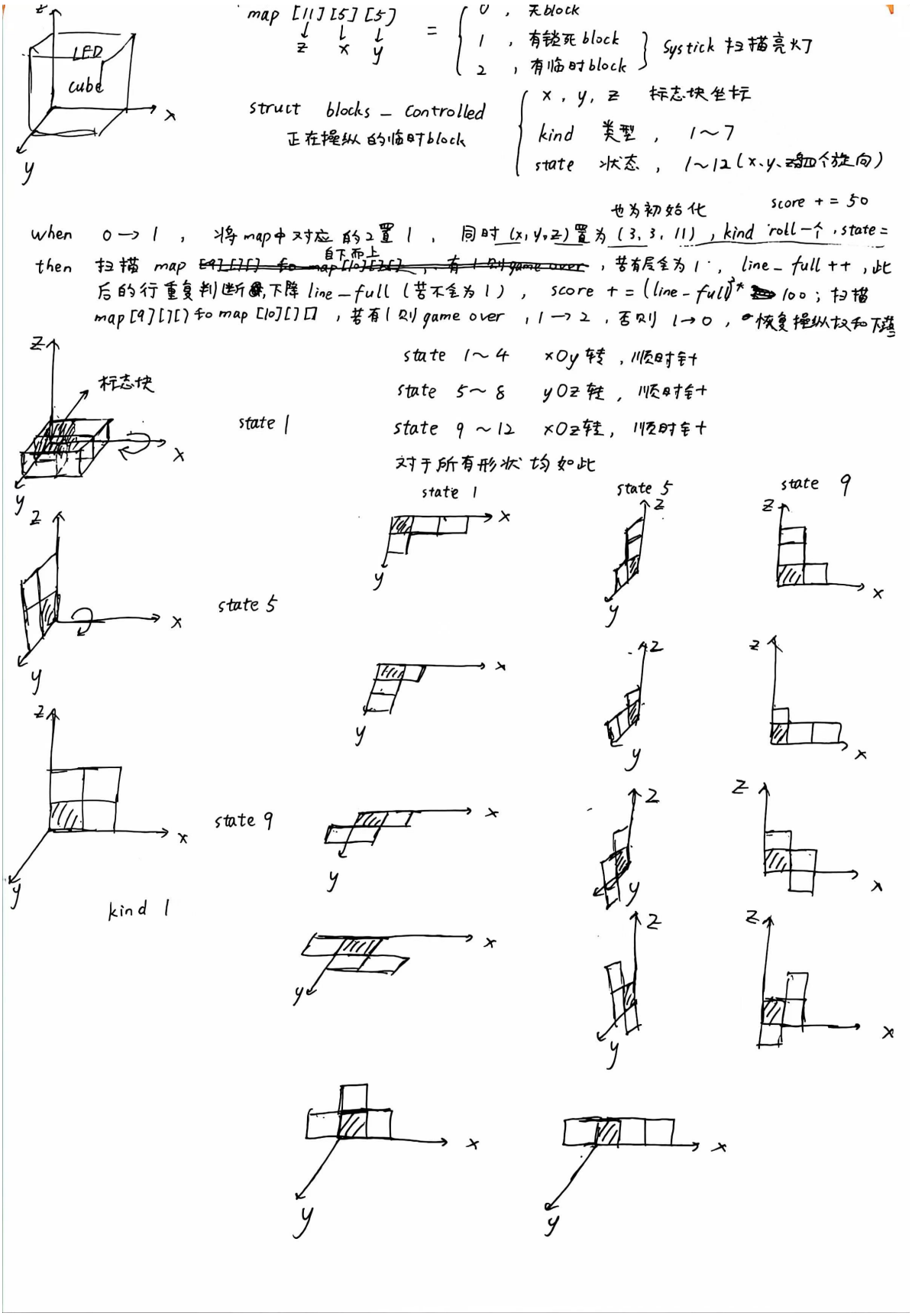
如上图所示，整个系统运行分为三个状态

1. 初始状态
2. 判断状态
3. 结算状态

虽然称不上状态机，但也可以捋清思路

可以先到main函数看一眼

主体逻辑实现



上面的图有些凌乱，细分来说：

方块的表示

为了方便读写分离，建立数组map[11][5][5]表示方块，也即map[z][x][y]的值表示是否存在方块、以及存在哪种方块：0表示无方块，1表示有锁死方块，2表示有临时方块

这样做将锁死方块和临时方块分开，便于处理。

临时方块的建立

可操纵的临时方块是这样的，锁死方块只需要置1并扫描就可以了，临时方块要考虑的可就多了。

三个要素：位置、形状、方向，我们不妨用一个结构体来存储信息。

怎么标定?

对于5种不同的形状（原本是7种，去掉了反L形和反Z形，原因后面会讲），各确定一个块作为标志位，将这个块的坐标(x,y,z)记入结构体来标定位置。

形状用一个整数表示即可，同时这样方便roll随机数。

方向就很值得聊了，我们用整数state表示12个方向，取值范围为1~12，具体规定如上图。

这样。我们的结构体blocks_controlled就建立完毕了，顺手再新建一个全局变量Temporary，由于我们可以通过更新这个变量来实现刷新新的块，所以整个程序我们只需要这一个变量来表示临时方块

```
struct blocks_controlled
{
  uint8_t x;
  uint8_t y;
  uint8_t z;//(x,y,z)
  uint8_t kind;//1~5
  uint8_t state;//1~12
};

extern struct blocks_controlled Temporary;
```

临时方块的操纵

操纵临时方块的函数全部在block.h中定义，在block.c中实现

```
void SET_Temporary();

void Temporary_On();
void Temporary_Off();
//void Block_On(uint8_t x, uint8_t y, uint8_t z);
//void Block_Off(uint8_t x, uint8_t y, uint8_t z);

void Temporary_forward();
void Temporary_backward();
void Temporary_left();
void Temporary_right();
void Temporary_rotate();
void Temporary_down();

//void Temporary_auto_down();
//use systick instead

uint8_t Block_or_not();
uint8_t Lock_or_not();
void Lock_Temporary();

void Eliminate();
void Fail_or_not();
```

SET_Temporary()用于刷新临时方块，在游戏开始时触发一次，此后每次由状态1回到状态0时触发一次。

方块种类用随机数抽取，种子用系统时产生。

```
void SET_Temporary()
{
  Temporary.x = 2;
  Temporary.y = 2;
  Temporary.z = 10;
  Temporary.state = 1;

  srand(HAL_GetTick());
  Temporary.kind = rand()%5 + 1;
}
```

Temporary_On()用于向map写入临时方块信息，在得到坐标、形状和方向后转化为map中保存的亮灯信息，此处为屎山代码，用switch-case逐行实现。

下面展现冰山一角，对照关系在上一小节的图中。

这也是去掉两个形状的原因。

```
void Temporary_On()
{
  uint8_t x = Temporary.x;
  uint8_t y = Temporary.y;
  uint8_t z = Temporary.z;
  switch(Temporary.kind)
  {
    case 1:
      switch(Temporary.state)
      {
  case 1:
    map[z][x][y] = 2;map[z][x+1][y] = 2;map[z][x][y+1] = 2;map[z][x+1][y+1] = 2;break;
  case 2:
    map[z][x][y] = 2;map[z][x-1][y] = 2;map[z][x][y+1] = 2;map[z][x-1][y+1] = 2;break;
  case 3:
    map[z][x][y] = 2;map[z][x-1][y] = 2;map[z][x][y-1] = 2;map[z][x-1][y-1] = 2;break;
  case 4:
    map[z][x][y] = 2;map[z][x+1][y] = 2;map[z][x][y-1] = 2;map[z][x+1][y-1] = 2;break;
  case 5:
    map[z][x][y] = 2;map[z][x][y+1] = 2;map[z+1][x][y] = 2;map[z+1][x][y+1] = 2;break;
  case 6:
    map[z][x][y] = 2;map[z][x][y-1] = 2;map[z+1][x][y] = 2;map[z+1][x][y-1] = 2;break;
  case 7:
    map[z][x][y] = 2;map[z][x][y-1] = 2;map[z-1][x][y] = 2;map[z-1][x][y-1] = 2;break;
  case 8:
    map[z][x][y] = 2;map[z][x][y+1] = 2;map[z-1][x][y] = 2;map[z-1][x][y+1] = 2;break;
  case 9:
    map[z][x][y] = 2;map[z][x+1][y] = 2;map[z+1][x][y] = 2;map[z+1][x+1][y] = 2;break;
  case 10:
    map[z][x][y] = 2;map[z][x+1][y] = 2;map[z-1][x][y] = 2;map[z-1][x+1][y] = 2;break;
  case 11:
    map[z][x][y] = 2;map[z][x-1][y] = 2;map[z-1][x][y] = 2;map[z-1][x-1][y] = 2;break;
  case 12:
    map[z][x][y] = 2;map[z][x-1][y] = 2;map[z+1][x][y] = 2;map[z+1][x-1][y] = 2;break;
      }
      break;
```

同理Temporary_Off()用于覆盖map的临时方块信息

```
void Temporary_Off()
{
  uint8_t x = Temporary.x;
  uint8_t y = Temporary.y;
  uint8_t z = Temporary.z;
  switch(Temporary.kind)
  {
    case 1:
      switch(Temporary.state)
      {
      case 1:
  map[z][x][y] = 0;map[z][x+1][y] = 0;map[z][x][y+1] = 0;map[z][x+1][y+1] = 0;break;
      case 2:
  map[z][x][y] = 0;map[z][x-1][y] = 0;map[z][x][y+1] = 0;map[z][x-1][y+1] = 0;break;
      case 3:
  map[z][x][y] = 0;map[z][x-1][y] = 0;map[z][x][y-1] = 0;map[z][x-1][y-1] = 0;break;
      case 4:
  map[z][x][y] = 0;map[z][x+1][y] = 0;map[z][x][y-1] = 0;map[z][x+1][y-1] = 0;break;
      case 5:
  map[z][x][y] = 0;map[z][x][y+1] = 0;map[z+1][x][y] = 0;map[z+1][x][y+1] = 0;break;
      case 6:
  map[z][x][y] = 0;map[z][x][y-1] = 0;map[z+1][x][y] = 0;map[z+1][x][y-1] = 0;break;
      case 7:
  map[z][x][y] = 0;map[z][x][y-1] = 0;map[z-1][x][y] = 0;map[z-1][x][y-1] = 0;break;
      case 8:
  map[z][x][y] = 0;map[z][x][y+1] = 0;map[z-1][x][y] = 0;map[z-1][x][y+1] = 0;break;
      case 9:
  map[z][x][y] = 0;map[z][x+1][y] = 0;map[z+1][x][y] = 0;map[z+1][x+1][y] = 0;break;
      case 10:
  map[z][x][y] = 0;map[z][x+1][y] = 0;map[z-1][x][y] = 0;map[z-1][x+1][y] = 0;break;
      case 11:
  map[z][x][y] = 0;map[z][x-1][y] = 0;map[z-1][x][y] = 0;map[z-1][x-1][y] = 0;break;
      case 12:
  map[z][x][y] = 0;map[z][x-1][y] = 0;map[z+1][x][y] = 0;map[z+1][x-1][y] = 0;break;
      }
      break;
```

同理Lock_Temporary()用于将临时方块锁死

```
void Lock_Temporary()
{
  Temporary_Off();
  uint8_t x = Temporary.x;
  uint8_t y = Temporary.y;
  uint8_t z = Temporary.z;
  switch(Temporary.kind)
  {
    case 1:
      switch(Temporary.state)
      {
  case 1:
    map[z][x][y] = 1;map[z][x+1][y] = 1;map[z][x][y+1] = 1;map[z][x+1][y+1] = 1;break;
  case 2:
    map[z][x][y] = 1;map[z][x-1][y] = 1;map[z][x][y+1] = 1;map[z][x-1][y+1] = 1;break;
  case 3:
    map[z][x][y] = 1;map[z][x-1][y] = 1;map[z][x][y-1] = 1;map[z][x-1][y-1] = 1;break;
  case 4:
    map[z][x][y] = 1;map[z][x+1][y] = 1;map[z][x][y-1] = 1;map[z][x+1][y-1] = 1;break;
  case 5:
    map[z][x][y] = 1;map[z][x][y+1] = 1;map[z+1][x][y] = 1;map[z+1][x][y+1] = 1;break;
  case 6:
    map[z][x][y] = 1;map[z][x][y-1] = 1;map[z+1][x][y] = 1;map[z+1][x][y-1] = 1;break;
  case 7:
    map[z][x][y] = 1;map[z][x][y-1] = 1;map[z-1][x][y] = 1;map[z-1][x][y-1] = 1;break;
  case 8:
    map[z][x][y] = 1;map[z][x][y+1] = 1;map[z-1][x][y] = 1;map[z-1][x][y+1] = 1;break;
  case 9:
    map[z][x][y] = 1;map[z][x+1][y] = 1;map[z+1][x][y] = 1;map[z+1][x+1][y] = 1;break;
  case 10:
    map[z][x][y] = 1;map[z][x+1][y] = 1;map[z-1][x][y] = 1;map[z-1][x+1][y] = 1;break;
  case 11:
    map[z][x][y] = 1;map[z][x-1][y] = 1;map[z-1][x][y] = 1;map[z-1][x-1][y] = 1;break;
  case 12:
    map[z][x][y] = 1;map[z][x-1][y] = 1;map[z+1][x][y] = 1;map[z+1][x-1][y] = 1;break;
```

显然这三个函数基本结构体一模一样，至于为什么不合成一个函数，是为了代码可读性~~（此为谎言，就是没想到）~~

然后是最屎山的一段，碰撞检测，实现临时方块到达任何边界（无论是地图边界还是锁死方块边界）后无法继续前进。

对于两种边界都需要检测，这个函数本身无方向性，实际上是检验现在临时方块自身所在位置是否合理

至于这看起来不合理的代码，我们先按下不表

```
uint8_t Block_or_not()
{
  uint8_t x = Temporary.x;
  uint8_t y = Temporary.y;
  uint8_t z = Temporary.z;
  switch(Temporary.kind)
  {
    case 1:
      switch(Temporary.state)
      {
      case 1:
      if(z<0 || x<0 || y<0 || z>10 || x>3 || y>3
          || map[z][x][y] || map[z][x+1][y] || map[z][x][y+1] || map[z][x+1][y+1])return 1;
      break;
      case 2:
      if(z<0 || x<1 || y<0 || z>10 || x>4 || y>3 || map[z][x][y] || map[z][x-1][y] || map[z][x][y+1] || map[z][x-1][y+1
      break;
      case 3:
      if(z<0 || x<1 || y<1 || z>10 || x>4 || y>4 || map[z][x][y] || map[z][x-1][y] || map[z][x][y-1] || map[z][x-1][y-1
      break;
      case 4:
      if(z<0 || x<0 || y<0 || z>10 || x>3 || y>4 || map[z][x][y] || map[z][x+1][y] || map[z][x][y-1] || map[z][x+1][y-1
      break;
      case 5:
      if(z<0 || x<0 || y<0 || z>9 || x>4 || y>3 || map[z][x][y] || map[z][x][y+1] || map[z+1][x][y] || map[z+1][x][y+1]
      break;
      case 6:
      if(z<0 || x<0 || y<1 || z>9 || x>4 || y>4 || map[z][x][y] || map[z][x][y-1] || map[z+1][x][y] || map[z+1][x][y-1]
```

Lock_or_not检测临时方块是否触地并返回布尔值（用uint8_t代替），锁死只会在下降时触发

```
uint8_t Lock_or_not()
{
  Temporary_Off();
  Temporary.z--;
  if(Block_or_not())
    {
      Temporary.z++;
      Temporary_On();
      return 1;
    }
  else
    {
      Temporary.z++;
      Temporary_On();
      return 0;
    }
}
```

配合这个函数，上面的碰撞检测函数就看起来合理多了，先擦除Temporary在map中的值，避免自己干扰自己的碰撞检测；然后Temporary下降一格，做碰撞检测，得到结果后，再将Temporary移回并重新写入map

结果来看，Temporary没有改变，但得到了碰撞检测的返回值，后面的移动函数类似

以前移为例

```
void Temporary_forward()
{
  Temporary_Off();
  Temporary.y--;
  if(Block_or_not())
    {
      Temporary.y++;
      Temporary_On();
    }
  else
    Temporary_On();
}
```

擦除原位置，移动，判断，返回并重写或直接写入新位置

其中下降比较特殊

```
void Temporary_down()
{
  Temporary_Off();
  Temporary.z--;
  if(Block_or_not())
    {
      Temporary.z++;
      Lock_Temporary();
    }
  else
    {
      Temporary_On();
    }
}
```

检测到碰撞后，应该执行返回并锁死

而在锁死函数的最后，会改变游戏状态，由状态0进入状态1

```
  case 9:
    map[z][x][y] = 1;map[z+1][x][y] = 1;map[z+2][x][y] = 1;map[z-1][x][y] = 1;break;
  case 10:
    map[z][x][y] = 1;map[z][x+1][y] = 1;map[z][x+2][y] = 1;map[z][x-1][y] = 1;break;
  case 11:
    map[z][x][y] = 1;map[z-1][x][y] = 1;map[z-2][x][y] = 1;map[z+1][x][y] = 1;break;
  case 12:
    map[z][x][y] = 1;map[z][x-1][y] = 1;map[z][x-2][y] = 1;map[z][x+1][y] = 1;break;
       }
      break;
    }
  Game_State = 1;
}
```

消除和失败检测

在状态1，首要做的就是进行消除并计算得分，然后就要进行失败判断，以判断下一步进入状态0还是状态2

```
void Eliminate()
{
  if(Game_State != 1)
    return;
  uint8_t line_full = 0;
  uint8_t flag = 0;
  for(int i=0;i<11;i++)
  {
    flag = 0;
    for(int j=0;j<5;j++)
    {
      if(flag)break;
      for(int k=0;k<5;k++)
      {
        if(!map[i][j][k])
        {
        flag = 1;
        break;
        }
      }
    }
    if(flag)
      for(int j=0;j<5;j++)
        for(int k=0;k<5;k++)
          map[i-line_full][j][k] = map[i][j][k];
    else
      line_full++;
  }
  for(int i=10;i>10-line_full;i--)
    for(int j=0;j<5;j++)
      for(int k=0;k<5;k++)
        map[i][j][k] = 0;
  Score += 100*line_full*line_full;
  Auto_fall_time -= 500*line_full;
}
```

这一段逻辑看起来挺复杂，实际上~~也不简单~~很简单

第一步，依此扫描所有层，有空位说明不满，跳出循环并下移；没空位说明已满，让line_full加一

第二步，将最上方line_full层置为0

第三步，计算分数并更新自动下降时间间隔

```
void Fail_or_not()
{
  if(Game_State != 1)
    return;
  uint8_t flag = 0;
  for(int i=10;i>=9;i--)
  {
    if(flag)break;
    for(int j=0;j<5;j++)
     {
      if(flag)break;
      for(int k=0;k<5;k++)
        if(map[i][j][k] == 1)
        {
          flag = 1;
          break;
        }
     }
  }
  if(flag)Game_State = 2;
  else
  {
    Game_State = 0;
    SET_Temporary();
  }
}
```

失败判断就很简单了，直接扫描最上面两层，有锁死方块直接进入状态2，否则回到状态1

LED扫描和摇杆扫描

因为需要点亮的灯的信息用全局变量map保存，故LED扫描与其他逻辑语句解耦成为独立的板块

```
void LED_Scan()
{
 for(int i=10;i>=0;i--)
 {
  RCLK2_L;
  for(int t=0;t<16;t++)
  {
   SRCLK2_L;
   if(t == i)
   {
    SER2_H;
   }
   else
   {
    SER2_L;
   }
```

```
      SRCLK2_H;
    }
    RCLK2_H;

    uint8_t target[32] = {0};

    for(int j=0;j<5;j++)
      for(int k=0;k<5;k++)
      {
        if(map[i][j][k] == 1 || map[i][j][k] == 2)
          target[5*j+k] = 1;
      }

    RCLK1_L;
    for(int t=0;t<32;t++)
    {
      SRCLK1_L;
      if(target[t])
      {
        SER1_H;
      }
      else
      {
        SER1_L;
      }
      SRCLK1_H;
    }
    RCLK1_H;

    HAL_Delay(0);

  }
}
```

唯一的难点在于坐标和74HC595的控制对接，我们采取顺序输入信号，倒序接线的方式点灯

Roll_Scan()函数负责监视摇杆是否被触发，简单的按键检测，不再赘述

main函数

变量初始化和第一个Temporary的设置

```
KR_H;
RK_H;

for(int i=0;i<11;i++)
  for(int j=0;j<5;j++)
    for(int k=0;k<5;k++)
map[i][j][k] = 0;

Game_State = 0;
Score = 0;
Auto_fall_time = 5000;
forward_prev = 0;
backward_prev = 0;
left_prev = 0;
right_prev = 0;
rotate_prev = 0;
down_prev = 0;

SET_Temporary();
Temporary_On();
```

以及极其简化的while(1)

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    switch(Game_State)
    {
      case 0:
        LED_Scan();
        Roll_Scan();
        break;

      case 1:
        Eliminate();
        LED_Scan();
        //LCD_Score();
        Fail_or_not();
        break;

      case 2:
        //LCD_Over();
        HAL_Delay(5000);
        restart();
    }

}
/* USER CODE END 3 */
```

可以看到其实我们写了LCD的函数，可以实现显示分数和game over，但由于PCB设计的问题，能用的引脚极其有限，而且接线很有难度，故不在项目中展示，可以用大板单独演示

问题与解决

整个项目的过程简直是灾难

首先我们的PCB设计耗费时间较长，且设计很糟糕，初版甚至没接MOS管

再版一样问题多多，比如接74HC595的引脚并没有用排针引出、VCC和GND都没有排针引出，极大地增加了调试的难度

四个阳极74HC595顺序是乱的，导致对应坐标接线难度飙升，需要对着电路图一点一点接

然后代码一样浪费了很多时间，开始时发现阴极不受控制，检查了半天代码未发现问题，从硬件上重焊多次，亦未解决，通过逻辑分析仪发现时钟引脚电平不受控，却找不到问题所在

最后发现

设置成了input引脚（）

白白浪费了大半天，还被Jerry狠狠吐槽

以及焊接也有问题，一些阳极和MOS管出焊接问题，重焊才解决

总之，这个项目确定方向很早，收尾却很晚，和我们的逆天操作不无关系

往好处想，至少学会了很多查错和调试技巧（）

希望大家引以为戒