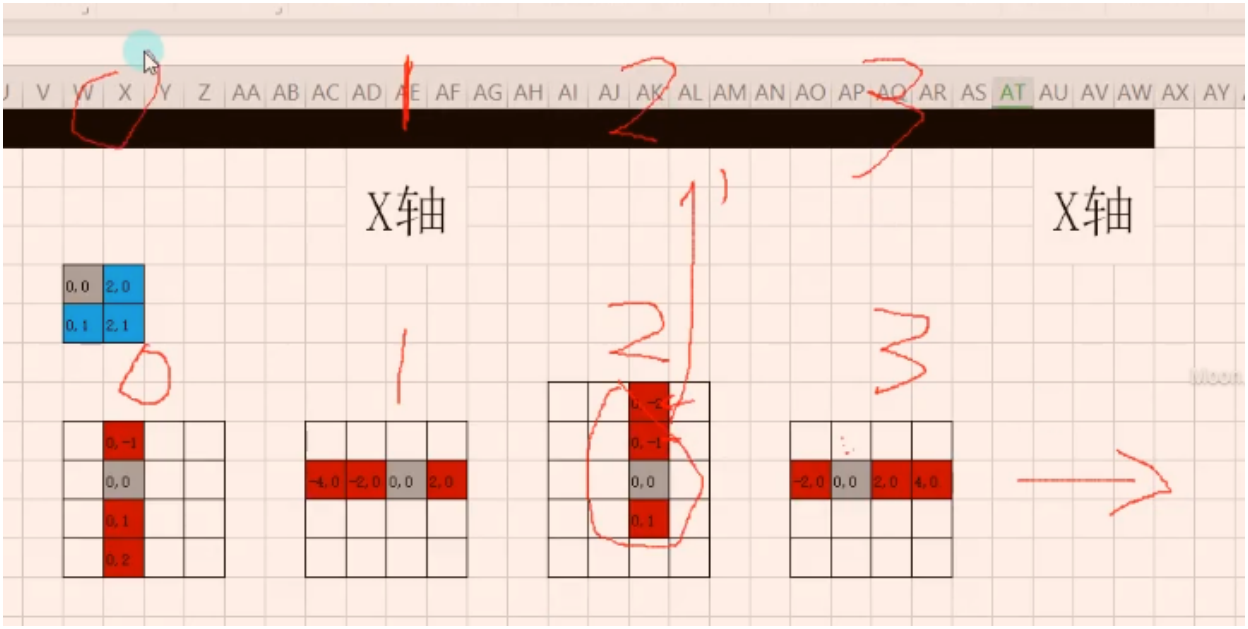
7：方块变形



也就是我们首先会随机到一个类型的方块，当我们按ad键的时候可以改变当前方块的形态，a往左变形，d就是往右变形，如果越界了，就设置为相反的极值：3->0　3<-0

```
using System.Text;

namespace 俄罗斯方块
{
    10 个引用
    enum E_ChangeType
    {
        Left,
        Right
    }

    3 个引用
    class BlockWorker:IDraw
    {
        //方块们
        private List<DrawObject> blocks;
        //选择容器来记录各个方块的形态信息
        
        //存储方块的几种信息
        private Dictionary<E_DrawType, BlockInfo> blockInfoDic;

        //记录当前随机出来的方块形态信息
        private BlockInfo nowBlockInfo;

        private int nowIndex = 0;

        1 个引用
        public BlockWorker()
        {
            //初始化，砖块信息,后续使用直接取
            blockInfoDic = new Dictionary<E_DrawType, BlockInfo>()
            {
                {E_DrawType.Cube,new BlockInfo(E_DrawType.Cube) },
                {E_DrawType.Line,new BlockInfo(E_DrawType.Line) },
                {E_DrawType.Tank,new BlockInfo(E_DrawType.Tank) },
                {E_DrawType.Left_Ladder,new BlockInfo(E_DrawType.Left_Ladder) },
                {E_DrawType.Right_Ladder,new BlockInfo(E_DrawType.Right_Ladder) },
                {E_DrawType.Left_Long_Ladder,new BlockInfo(E_DrawType.Left_Long_Ladder) },
                {E_DrawType.Right_Long_Ladder,new BlockInfo(E_DrawType.Right_Long_Ladder) },
            };
            //创建一个方块
            RandomCreateBlock();
        }

        //绘制
        3 个引用
        public void Draw()
```

```
        {
            //绘制创建好的砖
            for (int i = 0; i < blocks.Count; i++)
            {
                blocks[i].Draw();
            }
        }

        /// <summary>
        /// 随机创建一个下落的方块
        /// </summary>
        1 个引用
        public void RandomCreateBlock()
        {
            //随机方块类型
            Random random = new Random();
            E_DrawType type = (E_DrawType)random.Next(1, 8);

            //每次新建一个砖块，就是4个小方形
            blocks = new List<DrawObject>()
            {
                new DrawObject(type),
                new DrawObject(type),
                new DrawObject(type),
                new DrawObject(type),
            };
            //需要初始化方块位置
            //原点位置，随机自定义 方块List中的0个就是我们的原点方块
            blocks[0].pos = new Position(24, 5);
            //其他三个方块的位置
            //需要取出当前方块的形态，在进行随机取出变形后的方块
            nowBlockInfo = blockInfoDic[type];
            //不能随机0,4因为Cube类型方块只有一个数组元素，其他的都有4个数组元素
            nowIndex = random.Next(0, nowBlockInfo.Count);//在当前类型下的方块中，随机一种变形后的状态
            Position[] nowPos =  nowBlockInfo[nowIndex];
            //计算另外3个小放行的位置，基于原点计算
            for (int i = 0; i < nowPos.Length; i++)
            {
                //取出来的pos是独立的点，需要转换成相对原点的坐标
                blocks[i + 1].pos = nowPos[i] + blocks[0].pos;
            }
        }


        //擦除之前的方块
        1 个引用
        public void ClearDraw()
        {
            //调用最小单元 DrawObject的ClearDraw方法。封装简洁
            for (int i = 0; i < blocks.Count; i++)
            {
                blocks[i].ClearDraw();
            }
        }


        //变形相关方法
        2 个引用
        public void Change(E_ChangeType type)
        {
            //改变前，清理上一个形态
            ClearDraw();
            switch (type)
            {
                case E_ChangeType.Left://一共4个形态，下标 0~3 越界就取相反坐标的极值
                    //之所以不直接写 3 而是写 nowBlockInfo.Count-1 1是硬编码，后续有其他方块可能不是4个类型，
                    //2是因为Cube正方形 就只有一个类型:正方形，所以不能直接取 3来计算
                    nowIndex = nowIndex == 0 ? nowBlockInfo.Count-1 : --nowIndex;
                    break;
                case E_ChangeType.Right:
                    nowIndex = nowIndex == nowBlockInfo.Count - 1 ? 0 : ++nowIndex;
                    break;
            }
            //拿到新的形态后，开始变形
            Position[] nowPos = nowBlockInfo[nowIndex];//根据上面获取到的形态索引，取转换形态
            //计算另外3个小放行的位置，基于原点计算
            for (int i = 0; i < nowPos.Length; i++)
            {
                blocks[i + 1].pos = nowPos[i] + blocks[0].pos;
            }
            //在绘制
            Draw();
        }

        /// <summary>
        /// 判断是否进行变形
        /// 如果有墙体阻挡，就无法变形
        /// </summary>
        /// <param name="type">变形方向</param>
        /// <param name="map">地图信息</param>
        /// <returns></returns>
        2 个引用
        public bool CanChange(E_ChangeType type,Map map)
```

```
    public bool CanChange(E_ChangeType type,Map map)
    {

        //复用Change方法的部分代码

        //为了防止变形失败的出现，不能直接使用全局的nowIndex来获取
        //因为失败了的话，就不应该改变nowIndex
        int tmpIndex = nowIndex;

        switch (type)
        {
            case E_ChangeType.Left://一共4个形态，下标 0~3 越界就取相反坐标的极值
                //之所以不直接写 3 而是写 nowBlockInfo.Count-1 1是硬编码，后续有其他方块可能不是4个类型，
                //2是因为Cube正方形 就只有一个类型:正方形，所以不能直接取 3来计算
                tmpIndex = tmpIndex == 0 ? nowBlockInfo.Count - 1 : --tmpIndex;
                break;
            case E_ChangeType.Right:
                tmpIndex = tmpIndex == nowBlockInfo.Count - 1 ? 0 : ++tmpIndex;
                break;
        }
        //拿到新的形态后，开始变形
        Position[] nowPos = nowBlockInfo[tmpIndex];//通过临时索引找到这个索引形态是否能变形
        //1、判断是否超过墙体边界
        Position pos;
        for (int i = 0; i < nowPos.Length; i++)
        {
            pos = nowPos[i] + blocks[0].pos;
            if(pos.x<2||pos.x>=Game.w-2||
                    pos.y < 2 || pos.y >= map.h)
            {
                return false;
            }

        }
        //2、判断是否和下面的动态墙体重合
        for (int i = 0; i < nowPos.Length; i++)
        {
            pos = nowPos[i] + blocks[0].pos;
            for (int j = 0; j < map.dynamicWalls.Count; j++)
            {
                //跟动态墙体重合就不可变形
                if (pos == map.dynamicWalls[i].pos)
                {
                    return false;
                }
            }
        }
        return true;
    }
}
```

判断是否可以变形：

```
2 个引用
public void upData()//游戏界面(主要的逻辑处理更新)
{
    //地图更新绘制//每一帧都绘制一次
    map.Draw();
    //绘制砖块
    woker.Draw();

    switch (Console.ReadKey(true).Key)
    {
        case ConsoleKey.A:
            //判断是否可变形
            if (woker.CanChange(E_ChangeType.Left,map))
            {
                woker.Change(E_ChangeType.Left);
            }
            break;
        case ConsoleKey.D:
            if (woker.CanChange(E_ChangeType.Right, map))
            {
                woker.Change(E_ChangeType.Right);
            }
            break;
    }
}
```

优化Map类:

```
4 个引用
class Map:IDraw
{
    //固定墙壁
    public List<DrawObject> walls = new List<DrawObject>();

    //动态墙壁
    public List<DrawObject> dynamicWalls = new List<DrawObject>();

    //为了外部好获取墙体的范围
    public int h;
    public int w;

    1 个引用
    public Map()
    {

        h = Game.h - 6;//优化代码
        for (int i = 0; i < Game.w; i+=2)//添加下面的墙壁
        {
            walls.Add(new DrawObject(E_DrawType.Wall, i, h));
        }

        for (int i = 0; i < h; i++)//添加左右的墙壁
        {
            walls.Add(new DrawObject(E_DrawType.Wall, 0, i));//添加左边
            walls.Add(new DrawObject(E_DrawType.Wall, Game.w-2,i));//添加右边
        }
    }
```

总结：

1、首先考虑如何变形

2、是否可以变形

1、变形就是根据我们的按键，这里是AD 为变形键，A往左变形，D往右变形，从代码上讲，就是A 下标--，D下标++，如果越界了，就相反的极值。获取到下标的形态方块后，我们通过下标获取到里面的位置数组，将每一个位置元素根据原点位置进行相加，获取到这个形态的最终位置，然后调用最小单元的方块绘制方法，进行绘制。

在绘制的时候需要先判断是否可以绘制，是否可以绘制就是模拟当前已经按下的流程取获取下标，并模拟变形后的位置跟墙体和动态墙体进行比较是否有重合的，一旦有就不能变形