Android自定义View，九宫格解锁

史大拿 郭霖 2022-09-23 08:00 发表于江苏

/ 今日科技快讯 /

近日微软中国在其官方微博宣布，未来一年内将继续扩大在华招聘，员工总数预计突破1万人。在扩大招聘的基础上，微软还计划在未来三到五年内，对位于北京、上海及苏州的园区进行升级扩建，这三地园区的建设和运营将迎合当下混合式未来办公的灵活需求。据悉，微软中国现阶段正在实行混合工作制，员工超过50%的工作时间可居家办公。

/ 作者简介 /

明天就是周六啦，祝大家周末愉快！

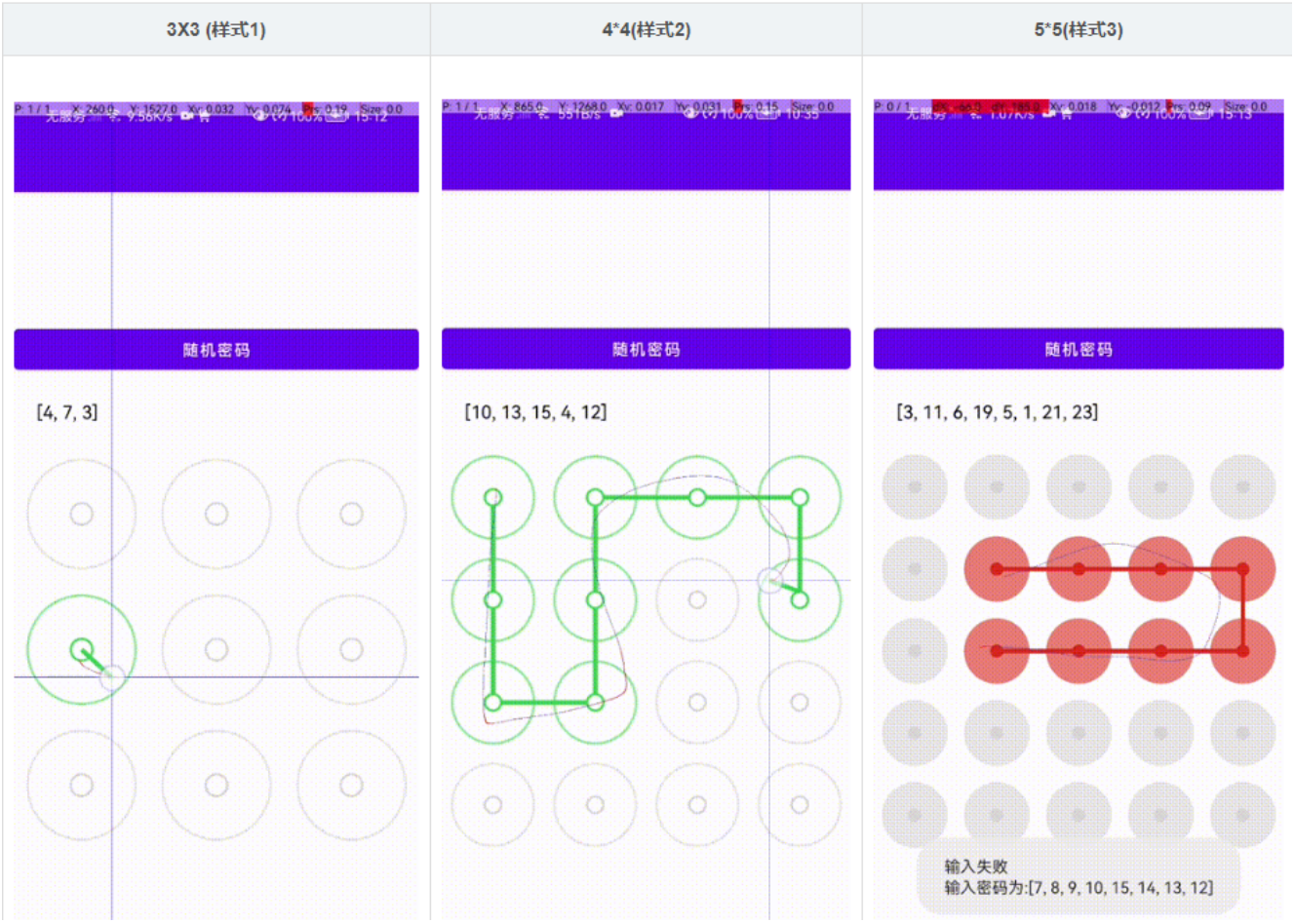
本篇文章来自史大拿的投稿，文章主要分享了手势解锁的实现，相信会对大家有所帮助！同时也感谢作者贡献的精彩文章。

史大拿的博客地址：

> https://blog.csdn.net/weixin_44819566?type=blog

/ 前言 /

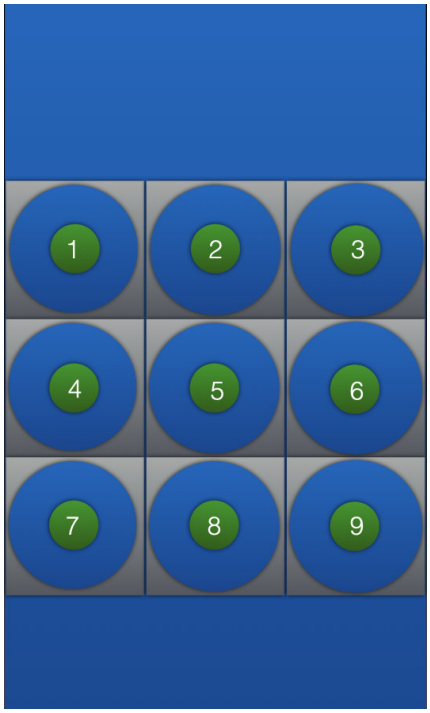
废话不多说,先来看今天要完成的效果:



Tips:不止3X3 或者 5X5 ,如果你想,甚至可以设置10*10

/ 画圆 /

先以3*3的九宫格来介绍!



我们要画成这样的效果, 画的是有一点丑,但是没关系.

首先来分析一下怎么花,这9个点的位置如何确定:

- 我们为了平均分, 单个圆的外层矩形 宽 = view.width / 3
- 高 = 宽
- 1号圆的圆心位置 = 0个矩形的宽度 = view.width / (3 * 2) + (view.width / 3) * 0
- 2号圆的圆心位置 = 1号圆的圆心位置 + 1个矩形的宽度 = view.width / (3 * 2) + (view.width / 3) * 1
- 3号圆的圆心位置 = 1号圆的圆心位置 + 2个矩形的宽度 = view.width / (3 * 2) + (view.width / 3) * 2

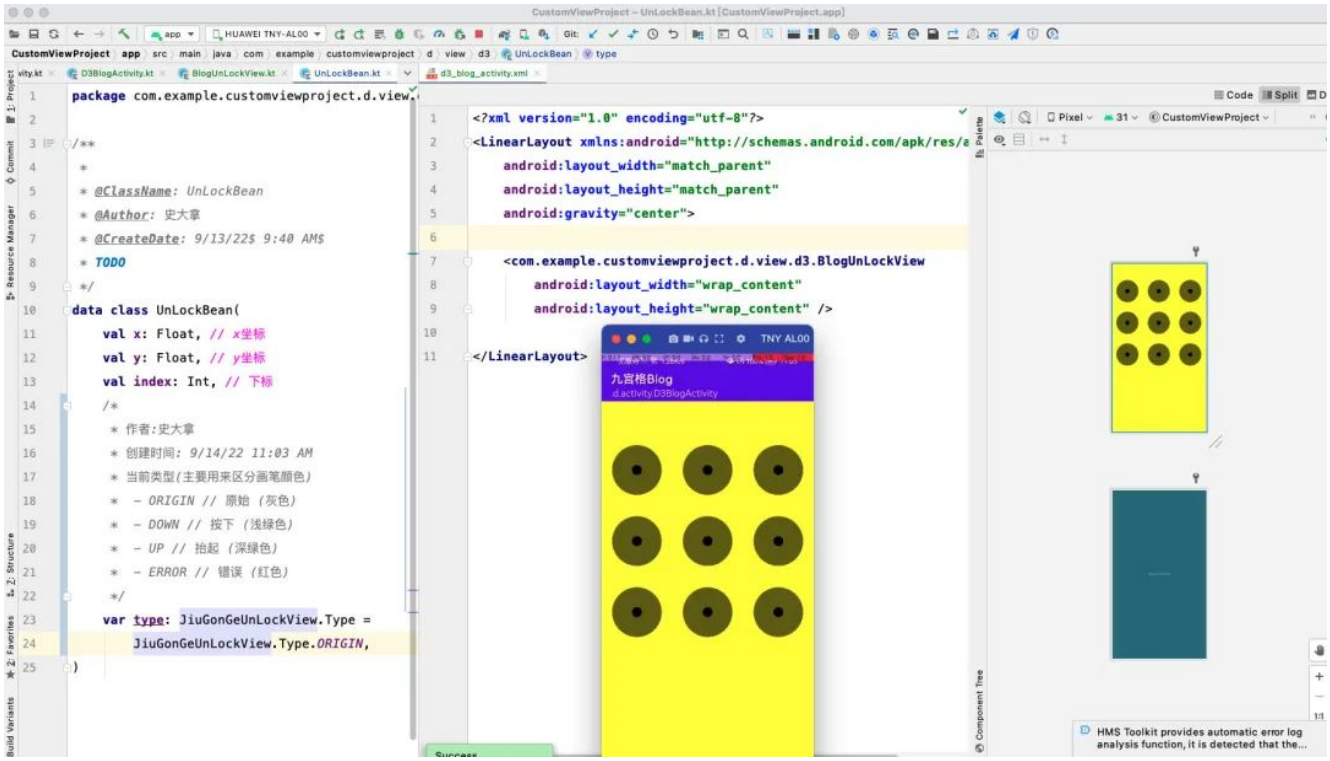
高坐标的计算也是如此

来看看目前的代码:

```
class BlogUnLockView @JvmOverloads constructor(
    context: Context, attrs: AttributeSet? = null, defStyleAttr: Int = 0
) : View(context, attrs, defStyleAttr) {
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
        //
        strokeJoin = Paint.Join.BEVEL
    }
```

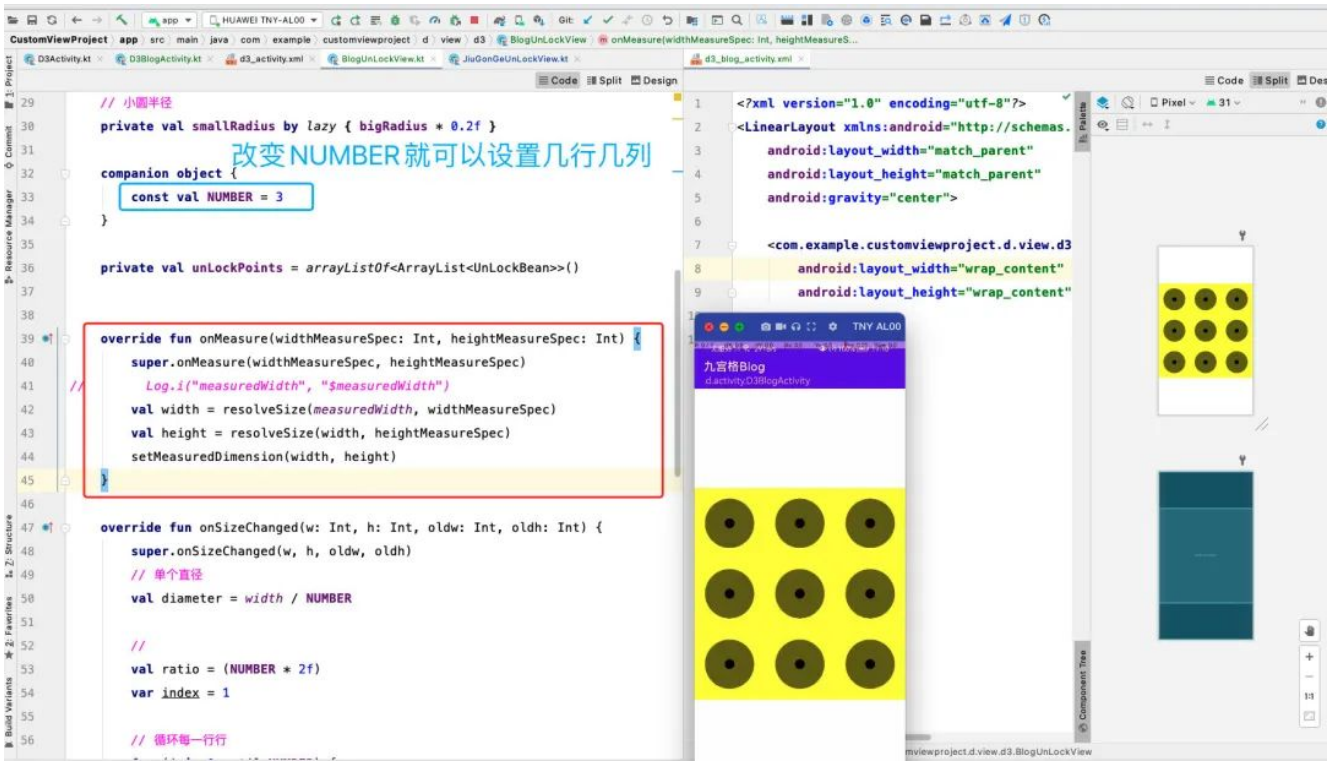
```
// 大圆半径
private val bigRadius by lazy { width / (NUMBER * 2) * 0.7f }

// 小圆半径
private val smallRadius by lazy { bigRadius * 0.2f }

companion object {
   const val NUMBER = 3
}

private val unLockPoints = arrayListOf<ArrayList<UnLockBean>>()

override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
   super.onSizeChanged(w, h, oldw, oldh)
   // 矩形直径
   val diameter = width / NUMBER

   //
   val ratio = (NUMBER * 2f)
   var index = 1

   // 循环每一行行
   for (i in 0 until NUMBER) {
      val list = arrayListOf<UnLockBean>()

      // 循环每一列
      for (j in 0 until NUMBER) {
         list.add(
            UnLockBean(
               width / ratio + diameter * j,
               height / ratio + diameter * i,
               index++
            )
         )
      }
      unLockPoints.add(list)
   }
}

override fun onDraw(canvas: Canvas) {
   canvas.drawColor(Color.YELLOW)

   unLockPoints.forEach {
      it.forEach { data ->
         // 绘制大圆
         paint.alpha = (255 * 0.6).toInt()
         canvas.drawCircle(data.x, data.y, bigRadius, paint)

         // 绘制小圆
```

```
                paint.alpha = 255
                canvas.drawCircle(data.x, data.y, smallRadius, paint)
            }
        }
    }
}
```

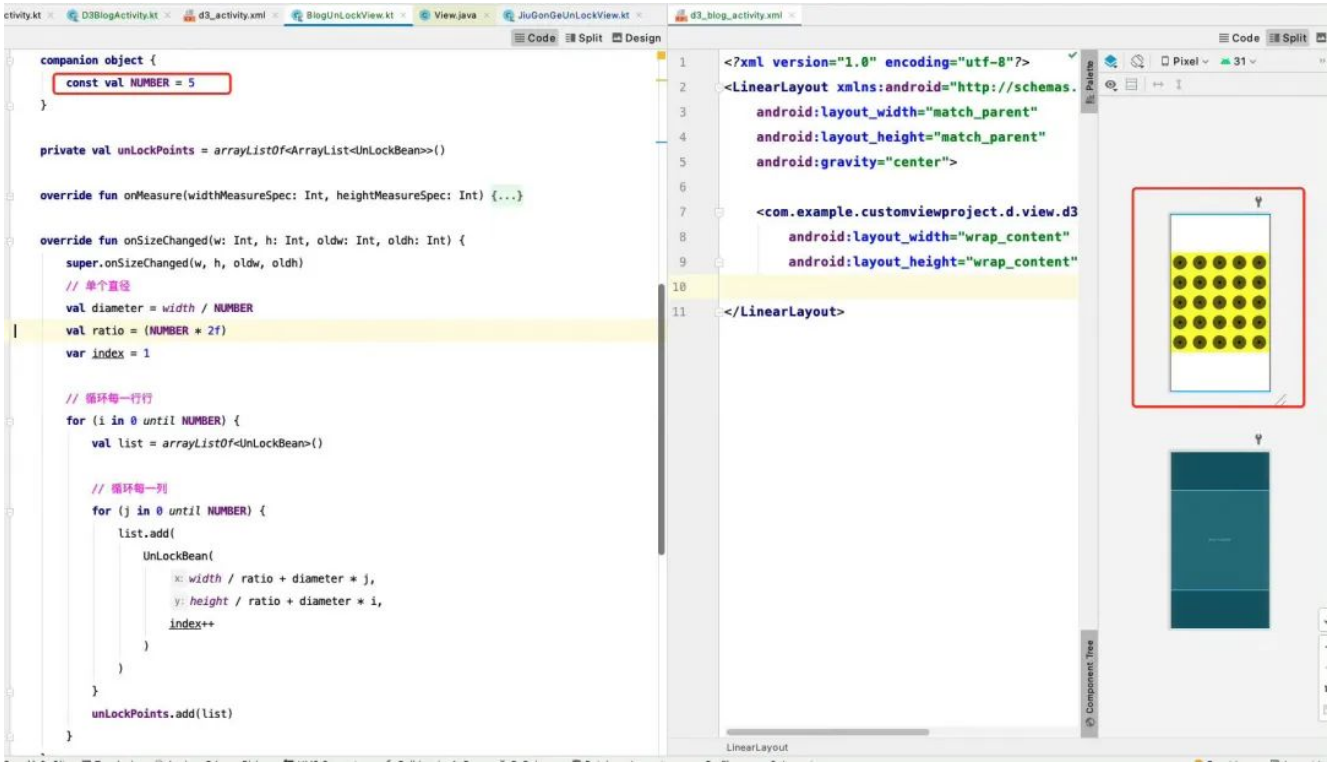


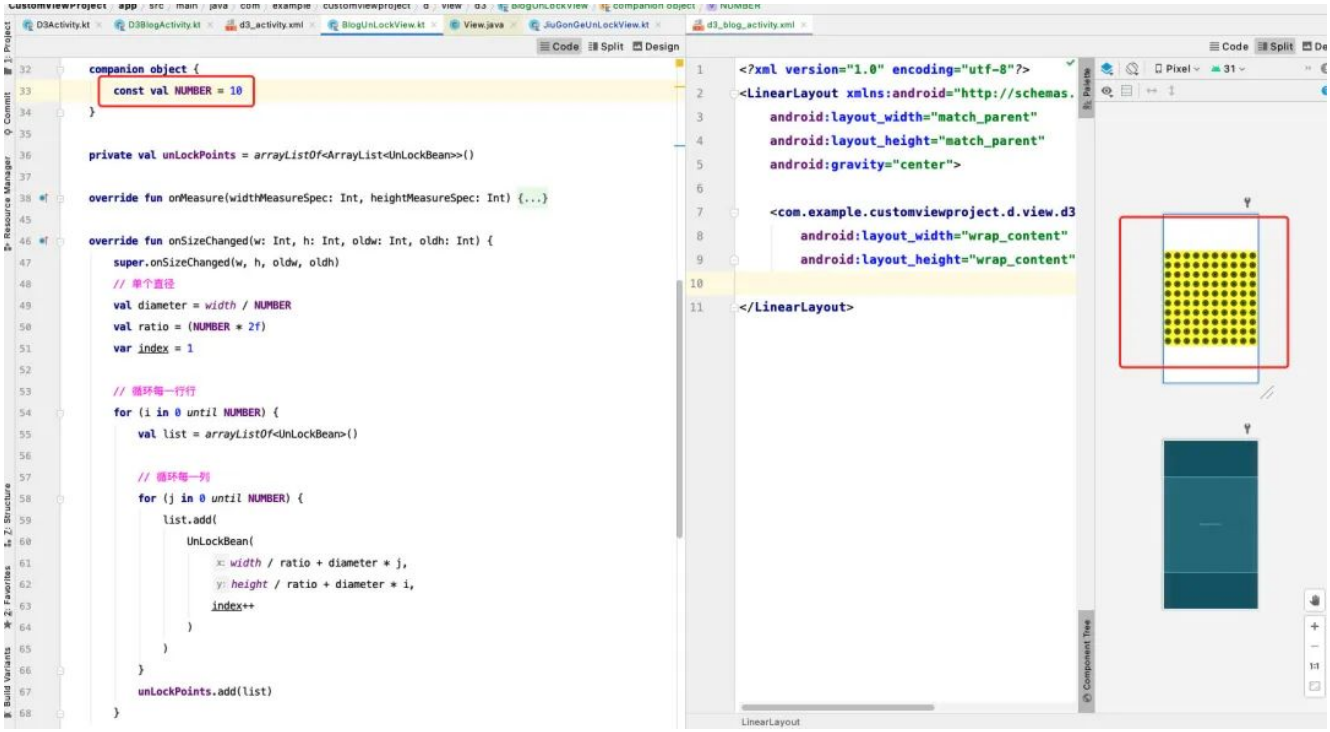
目前问题：整个view占满了屏幕,需要测量

测量代码比较简单,就是让宽和高一样即可



此时改变number变量,就可以设置几行几列，例如这样:





接下来我们就处理手势事件,按下滑动,抬起等,来改变选中

/ onTouchEvent /

在事件处理之前先来分析一下需要几种事件,对于解锁功能来说:

- ORIGIN 刚开始,还没有触摸
- DOWN 正在触摸中(输入密码)
- UP 触摸结束 (输入密码正确)
- ERROR 触摸结束 (输入密码错误)

那么就先定义4种颜色,来表示这4种状态:

```
companion object {

    // 原始颜色
    private var ORIGIN_COLOR = Color.parseColor("#D8D9D8")

    // 按下颜色
    private var DOWN_COLOR = Color.parseColor("#3AD94E")

    // 抬起颜色
    private var UP_COLOR = Color.parseColor("#57D900")
```

```
    // 错误颜色
    private var ERROR_COLOR = Color.parseColor("#D9251E")
}
```

接下来挨个处理事件

DOWN(按下)

首先需要思考,在按下的时候要做什么事情:

判断是否选中

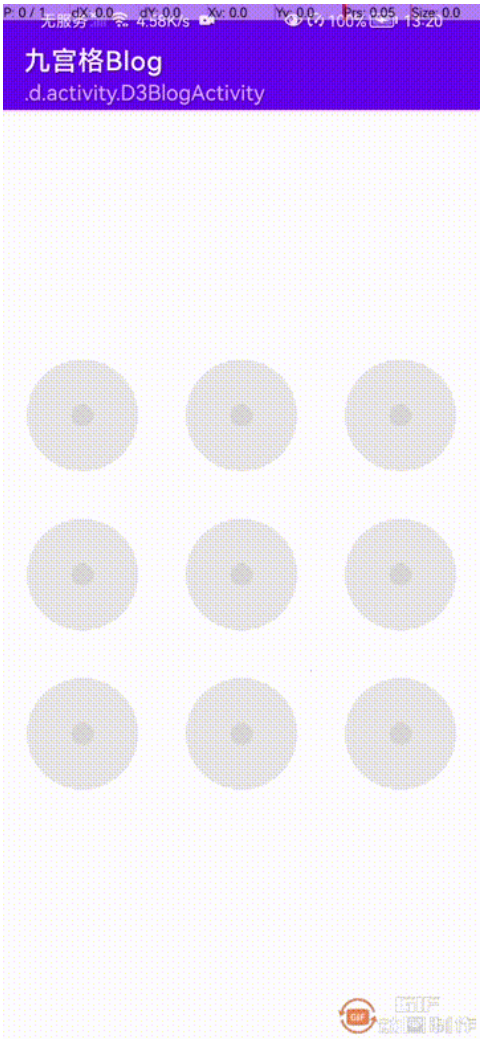
```
/*
* TODO 判断是否选中某个圆
* @param x,y: 点击坐标位置
*/
private fun isContains(x: Float, y: Float) = let {
    unLockPoints.forEach {
        it.forEach { data ->
            // 循环所有坐标 判断两个位置是否相同
            if (PointF(x, y).contains(PointF(data.x, data.y), bigRadius)) {
                return@let data
            }
        }
    }
    return@let null
}

// 判断一个点是否在另一个点范围内
fun PointF.contains(b: PointF, bPadding: Float = 0f): Boolean {
    val isX = this.x <= b.x + bPadding && this.x >= b.x - bPadding

    val isY = this.y <= b.y + bPadding && this.y >= b.y - bPadding
    return isX && isY
}
```

思路: 通过比较 按下位置和所有位置,判断是否有相同的

- 如果**有相同的**,那么就返回对应坐标
- 如果**没有相同的**,那么就返回null

```
@SuppressLint("ClickableViewAccessibility")
    override fun onTouchEvent(event: MotionEvent): Boolean {
```

```
        when (event.action) {
            MotionEvent.ACTION_DOWN -> {
                // 判断是否选中
                val pointF = isContains(event.x, event.y)
                pointF?.let {
                    // 将当前类型变为按下类型
                    it.type = JiuGonGeUnLockView.Type.DOWN
                }
            }
            ...
        }
        invalidate()
        return true
    }

override fun onDraw(canvas: Canvas) {
//     canvas.drawColor(Color.YELLOW)

        unLockPoints.forEach {
            it.forEach { data ->
                // 根据类型设置颜色
                paint.color = getTypeColor(data.type)

                // 绘制大圆
                paint.alpha = (255 * 0.6).toInt()
                canvas.drawCircle(data.x, data.y, bigRadius, paint)

                // 绘制小圆
                paint.alpha = 255
                canvas.drawCircle(data.x, data.y, smallRadius, paint)
            }
        }
}

/// TODO 获取类型对应颜色
private fun getTypeColor(type: JiuGonGeUnLockView.Type): Int {
    return when (type) {
        JiuGonGeUnLockView.Type.ORIGIN -> ORIGIN_COLOR
        JiuGonGeUnLockView.Type.DOWN -> DOWN_COLOR
        JiuGonGeUnLockView.Type.UP -> UP_COLOR
        JiuGonGeUnLockView.Type.ERROR -> ERROR_COLOR
    }
}
```

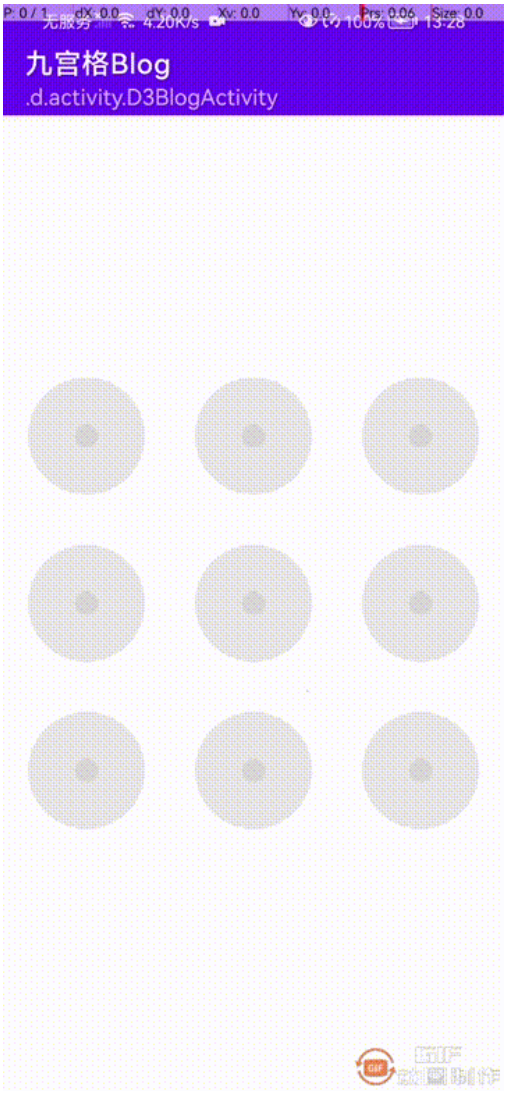


MOVE(移动)

move事件和down事件的逻辑是一样的,滑动的过程中判断点是否选中,然后绘制点

```
@SuppressLint("ClickableViewAccessibility")
override fun onTouchEvent(event: MotionEvent): Boolean {
    when (event.action) {
        MotionEvent.ACTION_DOWN -> {
            val pointF = isContains(event.x, event.y)
            pointF?.let {
                // 将当前类型改变为按下类型
                it.type = JiuGonGeUnLockView.Type.DOWN
            }
        }
        MotionEvent.ACTION_MOVE -> {
            val pointF = isContains(event.x, event.y)
            pointF?.let {
                // 将当前类型改变为按下类型
                it.type = JiuGonGeUnLockView.Type.DOWN
            }
        }
```

```
            ....
        }

        invalidate()
        return true
    }
```



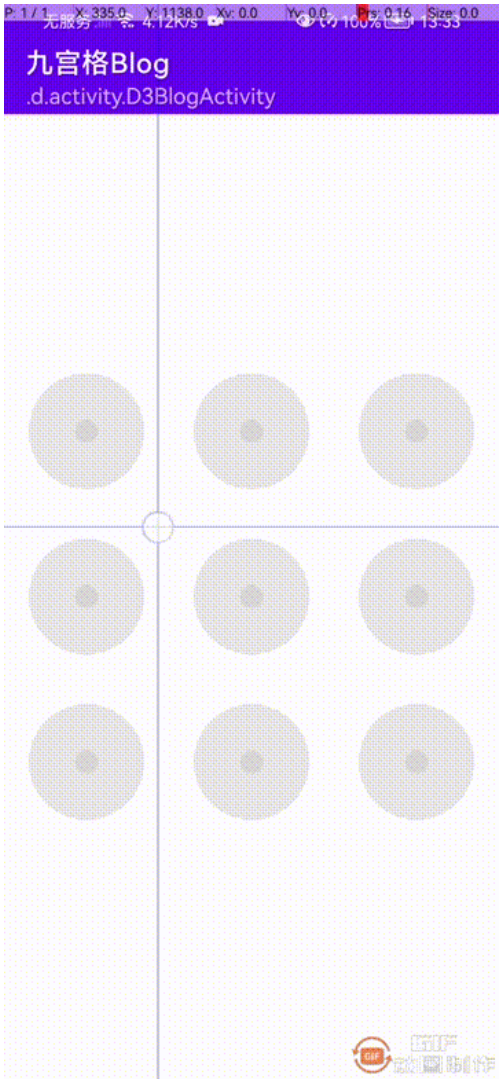
可以看出,效果是基本完成了,但是还有一个小错误

通常我们在九宫格的时候,一般都是先按下一个点才能滑动, 否则是不能滑动的,

现在的问题是,直接就可以滑动,所以还需要调整一下

那么我们就需要在down事件中标记一下是否按下,然后在move事件中判断一下

```
// 是否按下
private var isDOWN = false

@SuppressLint("ClickableViewAccessibility")
override fun onTouchEvent(event: MotionEvent): Boolean {
   when (event.action) {
      MotionEvent.ACTION_DOWN -> {
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            // 将当前类型改变为按下类型
            it.type = JiuGonGeUnLockView.Type.DOWN
            isDOWN = true // 表示按下
         }
      }
      MotionEvent.ACTION_MOVE -> {
         if (!isDOWN) {
            return super.onTouchEvent(event)
         }
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            // 将当前类型改变为按下类型
            it.type = JiuGonGeUnLockView.Type.DOWN
         }
      }

      MotionEvent.ACTION_CANCEL,
      MotionEvent.ACTION_UP -> {
         isDOWN = false // 标记没有按下
      }
   }

   invalidate()
   return true
}
```



UP(抬起)

思路分析:

抬起的时候要做很多事情

- 判断输入密码是否正确
  - 密码输入正确,那么就改变为深绿色
  - 密码输入错误,就改变为红色
- 完成之后,还需要吧所有的状态清空

在这里的时候,先不判断密码是否成功, 默认都是成功的,

- 先吧输入的密码toast出来

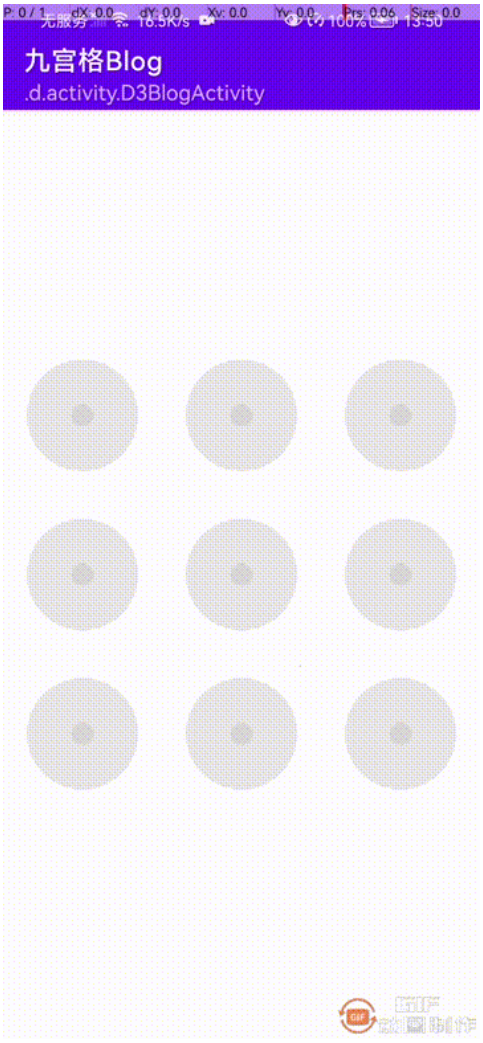
- 并且吧状态清空

等结尾的时候再来判断密码.

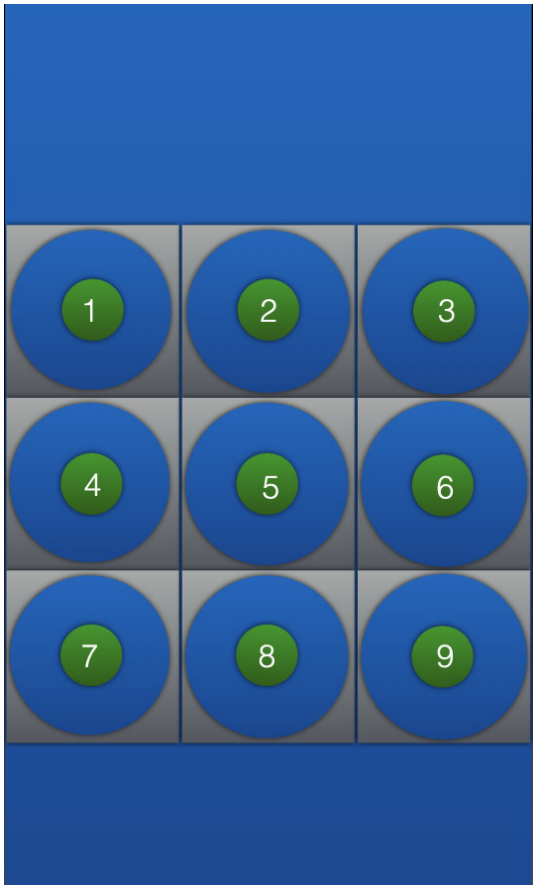
那么此时肯定是需要将所有选中的都记录下来, 然后在up事件中操作即可

```
// 记录选中的坐标
private val recordList = arrayListOf<UnLockBean>()

@SuppressLint("ClickableViewAccessibility")
override fun onTouchEvent(event: MotionEvent): Boolean {
   when (event.action) {
      MotionEvent.ACTION_DOWN -> {
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            // 将当前类型改变为按下类型
            it.type = JiuGonGeUnLockView.Type.DOWN
            isDOWN = true

            recordList.add(it)
         }
      }
      MotionEvent.ACTION_MOVE -> {
         if (!isDOWN) {
            return super.onTouchEvent(event)
         }
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            // 将当前类型改变为按下类型
            it.type = JiuGonGeUnLockView.Type.DOWN

            // 这里会重复调用，所以需要判断是否包含，如果不包含才添加
            if (!recordList.contains(it)) {
               recordList.add(it)
            }
         }
      }

      MotionEvent.ACTION_CANCEL,
      MotionEvent.ACTION_UP -> {
         // 将结果打印
         recordList.map {
            it.index
         }.toList() toast context

         clear()
      }
   }
```

```
    invalidate()
    return true
}

/// 清空所有状态
private fun clear() {
    recordList.forEach {
        // 将所有选中状态还原
        it.type = JiuGonGeUnLockView.Type.ORIGIN
    }
    recordList.clear()
    isDOWN = false // 标记没有按下

    invalidate()
}
```



/ 画线连接 /



假设现在需要连接 1,5,6,9

那么可以通过Path()来画线

在DOWN事件中,通过moveTo()移动到1的位置

在MOVE事件中,通过lineTo()画5,6,9的位置 即可

```
private val path = Path()

@SuppressLint("ClickableViewAccessibility")
override fun onTouchEvent(event: MotionEvent): Boolean {
   when (event.action) {
      MotionEvent.ACTION_DOWN -> {
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            /// 隐藏部分代码

            path.moveTo(it.x, it.y)
         }
      }
      MotionEvent.ACTION_MOVE -> {
               val pointF = isContains(event.x, event.y)
```

```
            pointF?.let {
            /// 隐藏部分代码

                // 这里会重复调用，所以需要判断是否包含，如果不包含才添加
                if (!recordList.contains(it)) {
                    recordList.add(it)
                    path.lineTo(it.x, it.y) // 连接到移动的位置
                }
            }
        }

        MotionEvent.ACTION_CANCEL,
        MotionEvent.ACTION_UP -> {
            // 将结果打印
            recordList.map {
                it.index
            }.toList() toast context


            clear()
        }
    }

    invalidate()
    return true
}

/*
 * 作者:史大拿
 * 创建时间: 9/14/22 1:38 PM
 * TODO 用来清空标记
 */
private fun clear() {
  path.reset() // 重置

  recordList.forEach {
    // 将所有选中状态还原
    it.type = JiuGonGeUnLockView.Type.ORIGIN
  }
  recordList.clear()
  isDOWN = false // 标记没有按下
}

override fun onDraw(canvas: Canvas) {
  paint.style = Paint.Style.FILL
  unLockPoints.forEach {
    /// 隐藏部分代码
  }

  paint.style = Paint.Style.STROKE
```

```
  paint.strokeWidth = 4.dp
  paint.color = DOWN_COLOR // 默认按下颜色
  canvas.drawPath(path, paint)
}
```

可以看出,已经完成了画连接线,但是还缺少一条指示当前手指位置的线,

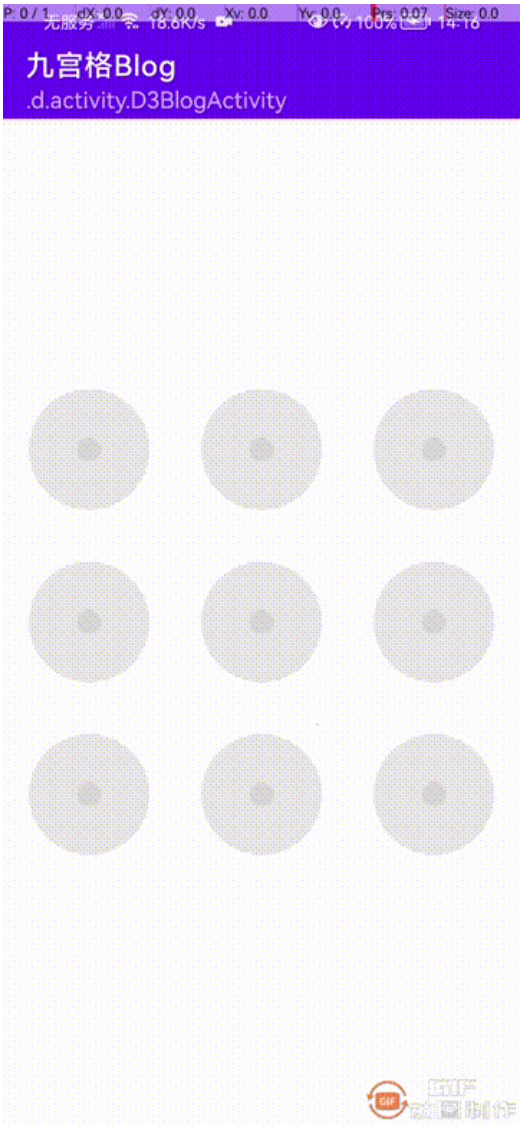
我叫他移动线, (好土的名字)

移动线就2个坐标

- 开始位置 (最后一个选中的位置)
- 结束位置 (当前手指按下的位置)

```
private val line = Pair(PointF(), PointF())

@SuppressLint("ClickableViewAccessibility")
override fun onTouchEvent(event: MotionEvent): Boolean {
   when (event.action) {
      MotionEvent.ACTION_DOWN -> {
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            /// 隐藏代码

            line.first.x = it.x
            line.first.y = it.y
         }
      }
      MotionEvent.ACTION_MOVE -> {
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            if (!recordList.contains(it)) {
                隐藏代码

               // 最后一个选中的位置
               line.first.x = it.x
               line.first.y = it.y
            }
         }

         // 手指的位置
         line.second.x = event.x
         line.second.y = event.y
      }
       ....
```

```
    }

    invalidate()
    return true
}

override fun onDraw(canvas: Canvas) {

    paint.style = Paint.Style.FILL
    unLockPoints.forEach {
            /// 隐藏代码
    }

        // 绘制连接线
    paint.style = Paint.Style.STROKE
    paint.strokeWidth = 4.dp
    paint.color = DOWN_COLOR // 默认按下颜色
    canvas.drawPath(path, paint)

    // 绘制移动线
    if (line.first.x != 0f && line.second.x != 0f
    ) {
      canvas.drawLine(
        line.first.x,
        line.first.y,
        line.second.x,
        line.second.y,
        paint
      )
    }
  }
```

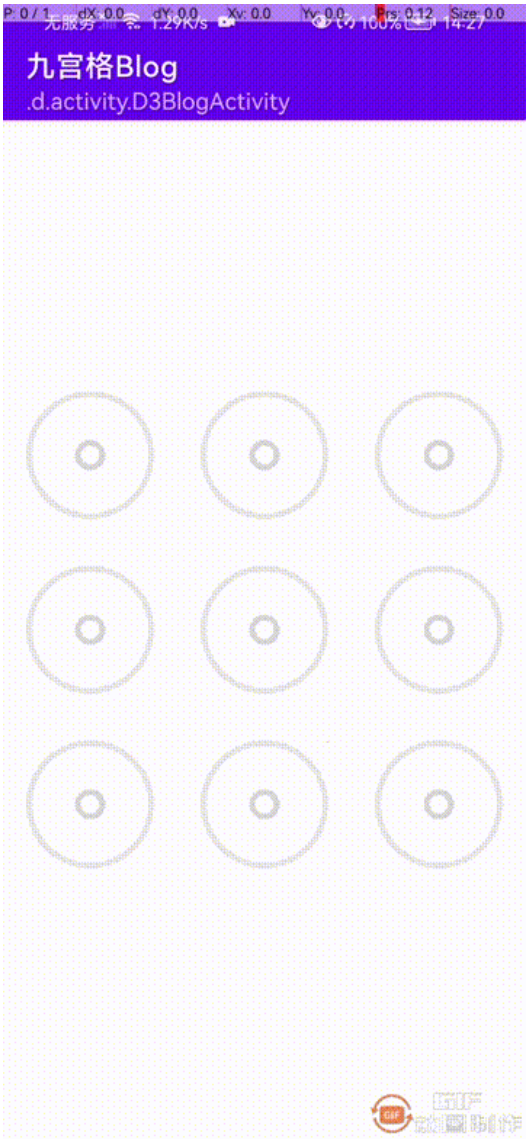


此时效果就差不多了,画笔默认是实心圆, 来看看空心效果

/ 空心效果 /

空心效果很简单,只需要调整画笔的style即可

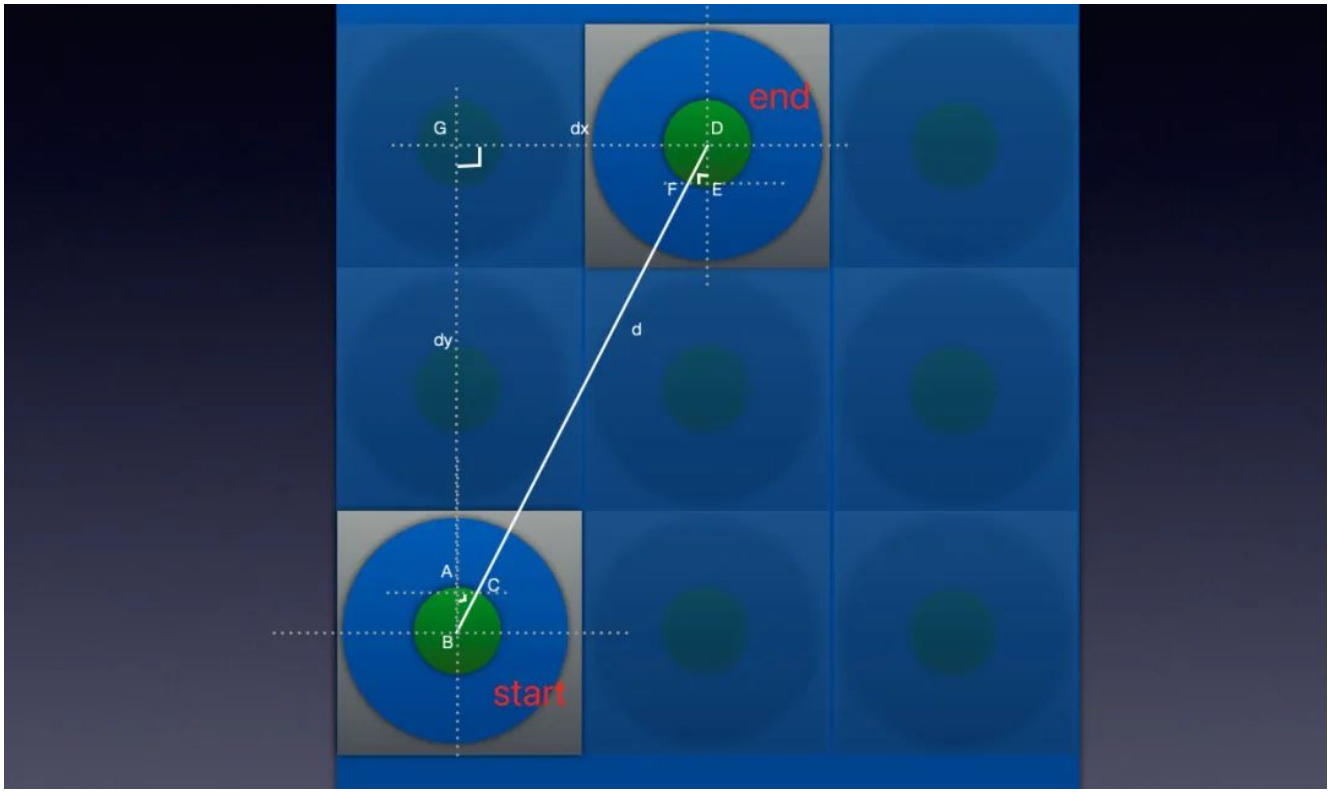
```
override fun onDraw(canvas: Canvas) {
  // 实心效果
//      paint.style = Paint.Style.FILL

  // 空心效果
     paint.style = Paint.Style.STROKE
     paint.strokeWidth = 4.dp

    // canvas.drawXXX()
  }
```



可以看出,此时的效果和我们想的一样,但是画线的时候从小圆圆心穿过了,不太好看

有没有一种办法,让线不从圆心穿过

那么就先来分析一下:



假设现在是从7移动到2

那么就需要连接C点和F点,只需要计算出C点和F点的坐标即可

先来分析现在的已知条件:

- dx = end.x - start.x
- dy = end.y - start.y
- d = (dx平方 + dy平方) 开根号
- 小圆半径 = smallRadius

那么就可以算出当前的偏移量:

- offsetX = dx * (smallRadius / d)
- offsetY = dy * (smallRadius / d)

知道偏移量,就可以算出C和F的坐标:

那么C的坐标为:

- C.x = start.x + offsetX
- C.y = start.y + offsetY

那么F的坐标为:

- F.x = end.x + offsetX
- F.y = end.y + offsetY

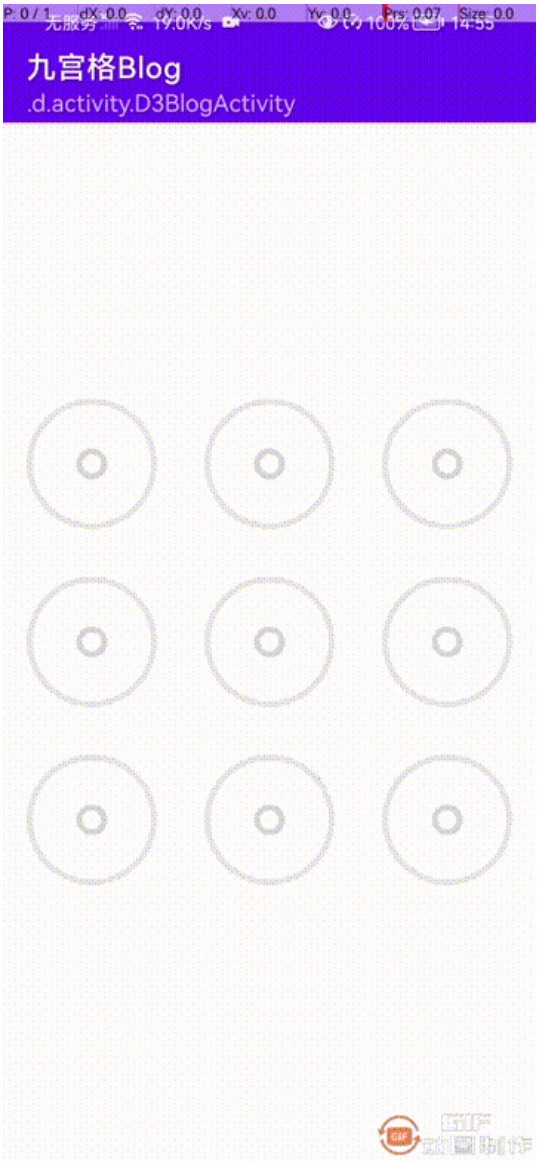
只要C和F的坐标之后

只需要通过path.moveTo() 移动到C的位置

通过path.lineTo() 移动到F的位置即可

```
@SuppressLint("ClickableViewAccessibility")
override fun onTouchEvent(event: MotionEvent): Boolean {
   when (event.action) {
      MotionEvent.ACTION_DOWN -> {
        /// ...
      }
      MotionEvent.ACTION_MOVE -> {
         val pointF = isContains(event.x, event.y)
         pointF?.let {
            // 将当前类型改变为按下类型
            it.type = JiuGonGeUnLockView.Type.DOWN

            // 这里会重复调用，所以需要判断是否包含，如果不包含才添加
            if (!recordList.contains(it)) {
               recordList.add(it)
               if (recordList.size >= 2) {
                  // TODO 不穿过圆心
                  val start = recordList[recordList.size - 2]
                  val end = recordList[recordList.size - 1]

                  val d = PointF(start.x, start.y).distance(PointF(end.x, end.y))
                  val dx = (end.x - start.x)
                  val dy = (end.y - start.y)
                  val offsetX = dx * smallRadius / d
                  val offsetY = dy * smallRadius / d

                  val cX = start.x + offsetX
                  val cY = start.y + offsetY
                  path.moveTo(cX, cY)
```

```
                val fX = end.x - offsetX
                val fY = end.y - offsetY
                path.lineTo(fX, fY)

                // line
                line.first.x = it.x + offsetX
                line.first.y = it.y + offsetY
            }
        }
    }

    // 手指的位置
    line.second.x = event.x
    line.second.y = event.y
  }

  /// 隐藏UP代码
 }

 invalidate()
 return true
}


// 计算两点之间的距离
fun PointF.distance(b: PointF): Float = let {
 val a = this

 // 这里 * 1.0 是为了转Double
 val dx = b.x - a.x * 1.0
 val dy = b.y - a.y * 1.0
 return@let sqrt(dx.pow(2) + dy.pow(2)).toFloat()
}
```



所有的效果基本就差不多了,接下来来比较密码

/ 比较密码 /

思路分析:

先将正确密码集合传过来,然后和输入的密码做比较

首先先判断两个集合的长度如果长度不一样,那么密码肯定是不同的,直接标记为错误即可

如果长度一样,只需要比较每一个值是否相同相同则输入成功,将正确结果回调回去

有一个不相同,则输入失败,标记为错误即可

```
// 密码
open var password = listOf<Int>()

MotionEvent.ACTION_UP -> {
   // 清空移动线
   line.first.x = 0f
   line.first.y = 0f
   line.second.x = 0f
   line.second.y = 0f


   // 标记是否成功
   val isSuccess =
         // 先比较长度是否相同
      if (recordList.size == password.size) {
         val list = recordList.zip(password).filter {
            // 通过判断每一个值
            it.first.index == it.second
         }.toList()

         // 如果每一个值都相同，那么就成功
         list.size == password.size
      } else {
         false
      }

   // 密码错误，将标记改变成成错误
   if (!isSuccess) {
      recordList.forEach {
         it.type = JiuGonGeUnLockView.Type.ERROR
      }
      "输入失败" toast context
   } else {
      "输入成功" toast context
   }

   // 延迟1秒清空
   postDelayed({
      clear()
   }, 1000)
}
```

现在已经可以完成输入密码了,

但是状态还不对,我们希望连接线的颜色和圆的颜色一致,

当然我们可以这样:

```
override fun onDraw(canvas: Canvas)
//      paint.style = Paint.Style.FILL
      paint.style = Paint.Style.STROKE
      paint.strokeWidth = 4.dp

      unLockPoints.forEach {
         it.forEach { data ->
```

```
            // 根据类型设置颜色
            paint.color = getTypeColor(data.type)

            // 绘制大圆
            paint.alpha = (255 * 0.6).toInt()
            canvas.drawCircle(data.x, data.y, bigRadius, paint)

            // 绘制小圆
            paint.alpha = 255
            canvas.drawCircle(data.x, data.y, smallRadius, paint)

                      // 绘制连接线
            canvas.drawPath(path, paint)

            // 绘制移动线
            if (line.first.x != 0f && line.second.x != 0f
            ) {
               canvas.drawLine(
                  line.first.x,
                  line.first.y,
                  line.second.x,
                  line.second.y,
                  paint
               )
            }
         }
      }
   }
```

但是我还是选择了通过一个全局变量,来记录当前的状态,然后给连接线和移动线设置颜色

代码很简单,就不展示了,直接看效果:

到此时,效果就基本完成了,

但是,写完发现,代码真的太乱了,而且有很多设置的东西,

比如说:

- 默认颜色
- 移动颜色
- 输入成功颜色
- 输入失败颜色
- 解锁的大小

- 例如3,就是3 X 3 5就是5 X 5
- 样式
- 空心 or 实心

一般遇到这种情况我认为有2种方式

- 自定义属性
- 设计模式

自定义属性用的很多,这里我就通过Adapter模式来优化一下

先来定义规范

```
abstract class UnLockBaseAdapter {
    // 设置宫格个数
    // 例如输入3: 表示3*3
    abstract fun getNumber(): Int

    // 设置样式
    abstract fun getStyle(): JiuGonGeUnLockView.Style

    /*
     * 作者:史大拿
     * 创建时间: 9/14/22 10:24 AM
     * TODO 画连接线时,是否穿过圆心
     */
    open fun lineCenterCircle() = false

    // 设置原始颜色
    open fun getOriginColor(): Int = let {
        return Color.parseColor("#D8D9D8")
    }

    // 设置按下颜色
    open fun getDownColor(): Int = let {
        return Color.parseColor("#3AD94E")
    }

    // 设置抬起颜色
    open fun getUpColor(): Int = let {
        return Color.parseColor("#57D900")
    }

    // 设置错误颜色
```

```
    open fun getErrorColor(): Int = let {
        return Color.parseColor("#D9251E")
    }
}
```

实现:

```
class UnLockAdapter : UnLockBaseAdapter() {
    override fun getNumber(): Int = 5

    override fun getStyle(): JiuGonGeUnLockView.Style = JiuGonGeUnLockView.Style.STROKE

    override fun getOriginColor(): Int {
        return Color.YELLOW
    }
}
```

读取数据:

```
open var adapter: UnLockBaseAdapter? = null

override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec)

    if (adapter == null) {
        throw AndroidRuntimeException("请设置Adapter")
    }
    adapter?.also {
        NUMBER = it.getNumber()
        ORIGIN_COLOR = it.getOriginColor()
        DOWN_COLOR = it.getDownColor()
        UP_COLOR = it.getUpColor()
        ERROR_COLOR = it.getErrorColor()
    }
}
```

来看看最终效果:

思路参考： https://www.jianshu.com/p/74e760ef8d10

完整代码：https://gitee.com/lanyangyangzzz/custom-view-project

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

AOP思想与插件化技术在安卓上的实践应用

谈一谈在两个商业项目中使用MVI架构后的感悟

欢迎关注我的公众号

学习技术或投稿

▼

长按上图，识别图中二维码即可关注

阅读原文

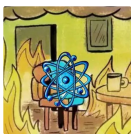
喜欢此内容的人还喜欢

React 可组合 API 的设计原则

KooFE前端团队

React 中文周刊 #109 - 深度解析 React

印记中文



狂肝半个月！1.3 万字深度剖析 Vue3 响应式（附脑图）

前端瓶子君