**欢迎关注微信公众号：孔乙己学习成长录**

**有任何问题，欢迎公众号后台提问，了解更多。**

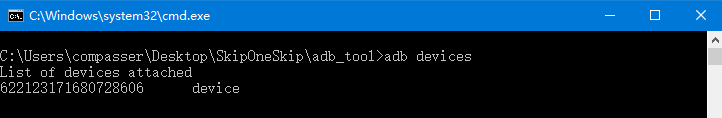
**跳一跳项目实训**

# 1、手机连接电脑：



# 2、安装对应的手机驱动程序：

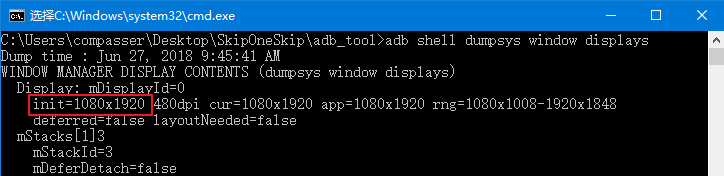
|  |
| --- |
| adb devices |



# 3、获取手机相关信息：

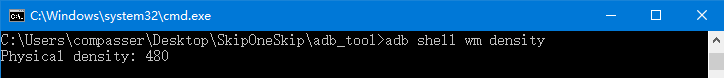
**（1）、查看手机分辨率**

|  |
| --- |
| adb shell dumpsys window displays |



**（2）、获取屏幕密度**

|  |
| --- |
| adb shell wm density |



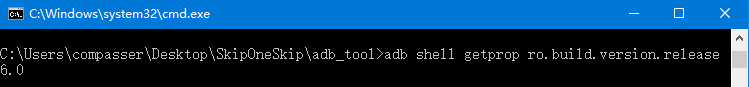
**（3）、获取手机型号**

|  |
| --- |
| adb shell getprop ro.product.device |



**（4）、获取Android系统的版本**

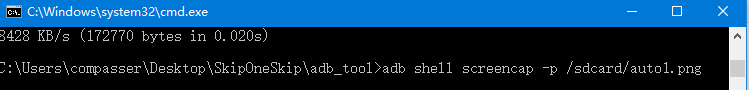
|  |
| --- |
| adb shell getprop ro.build.version.release |



# 4、截屏：

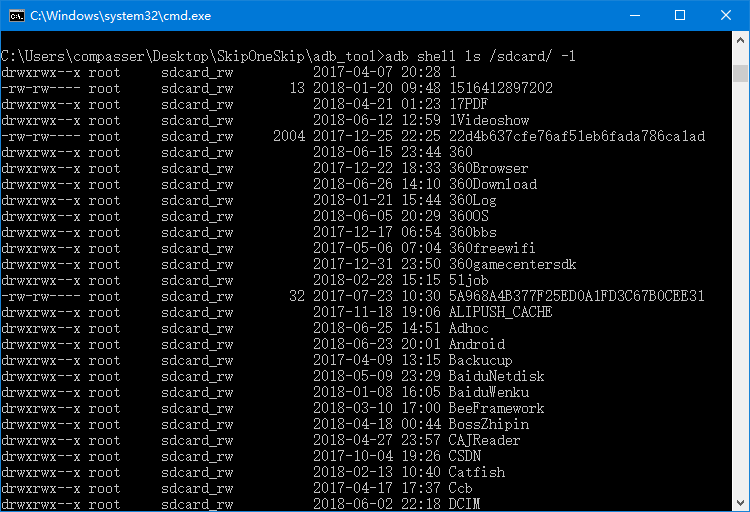
**（1）、截屏的图片就保存到 /sdcard/auto1.png文件中**

|  |
| --- |
| adb shell screencap -p /sdcard/auto1.png |
|  |



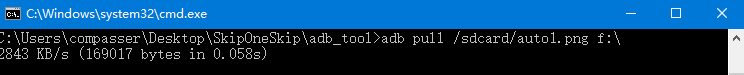
**（2）、查看sdcard目录下所有的文件**

|  |
| --- |
| adb shell ls /sdcard/ -l |



**（3）、把手机上的文件拷贝到电脑上**

|  |
| --- |
| adb pull /sdcard/auto1.png f:\ |



**（4）、打开即可看到所截取的当前手机屏幕信息：**



# 5、屏幕点击事件

**（1）、模拟手机滑动事件**

|  |
| --- |
| adb shell input swipe x1 y1 x2 y2 duration |

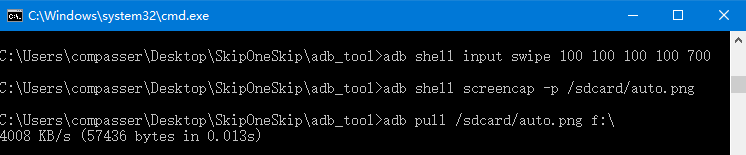
通过adb shell input swipe命令进行滑动

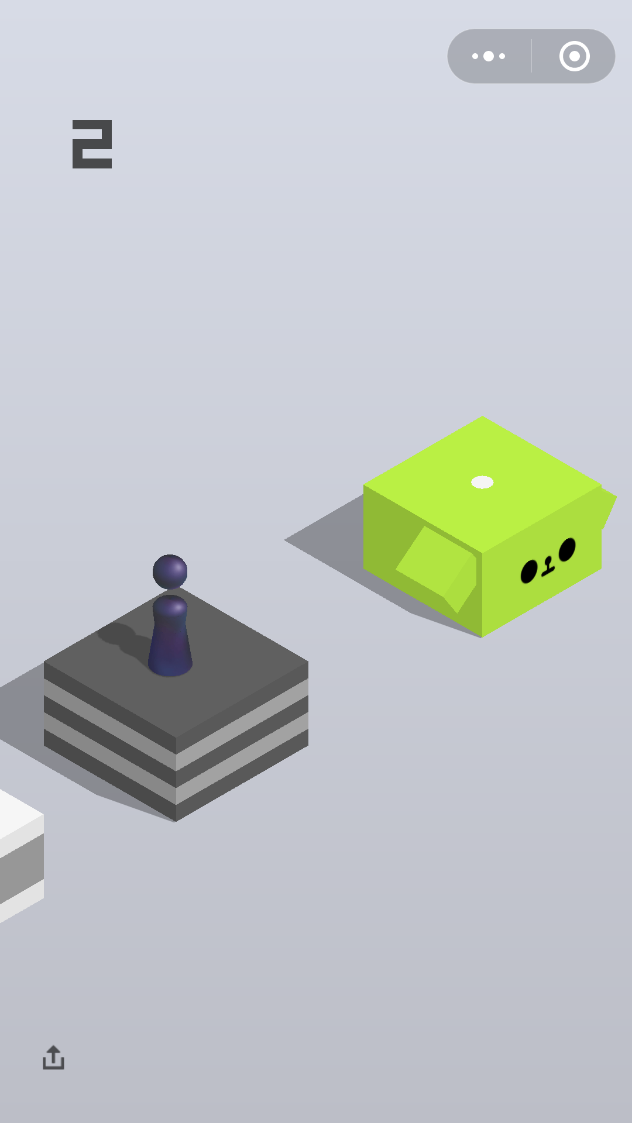
* x1、y1：滑动开始的点
* x2、y2：滑动结束的点
* duration：持续的时间（单位ms）

特殊情况下：如果不写duration参数，就理解为点击事件。如果写duration，然后x1y1和x2y2是相同的点，就表示长按。

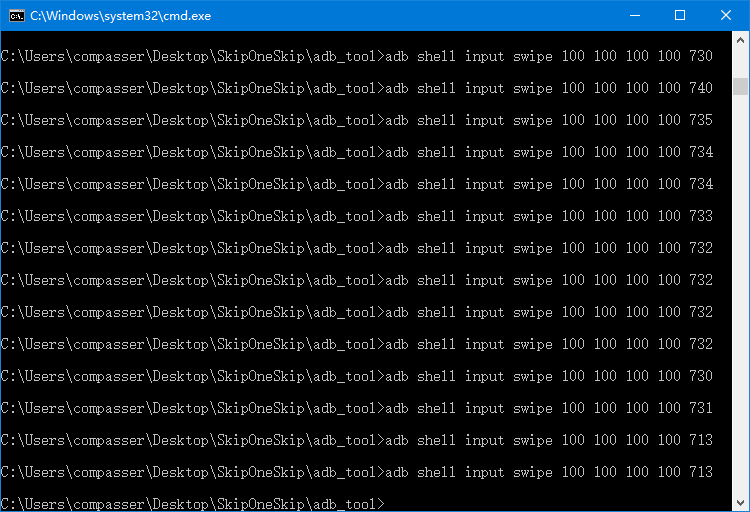
跳一跳关键是：duration的值的计算，例如，当duration 值为 700ms 使：

|  |
| --- |
| adb shell input swipe 100 100 100 100 700 |





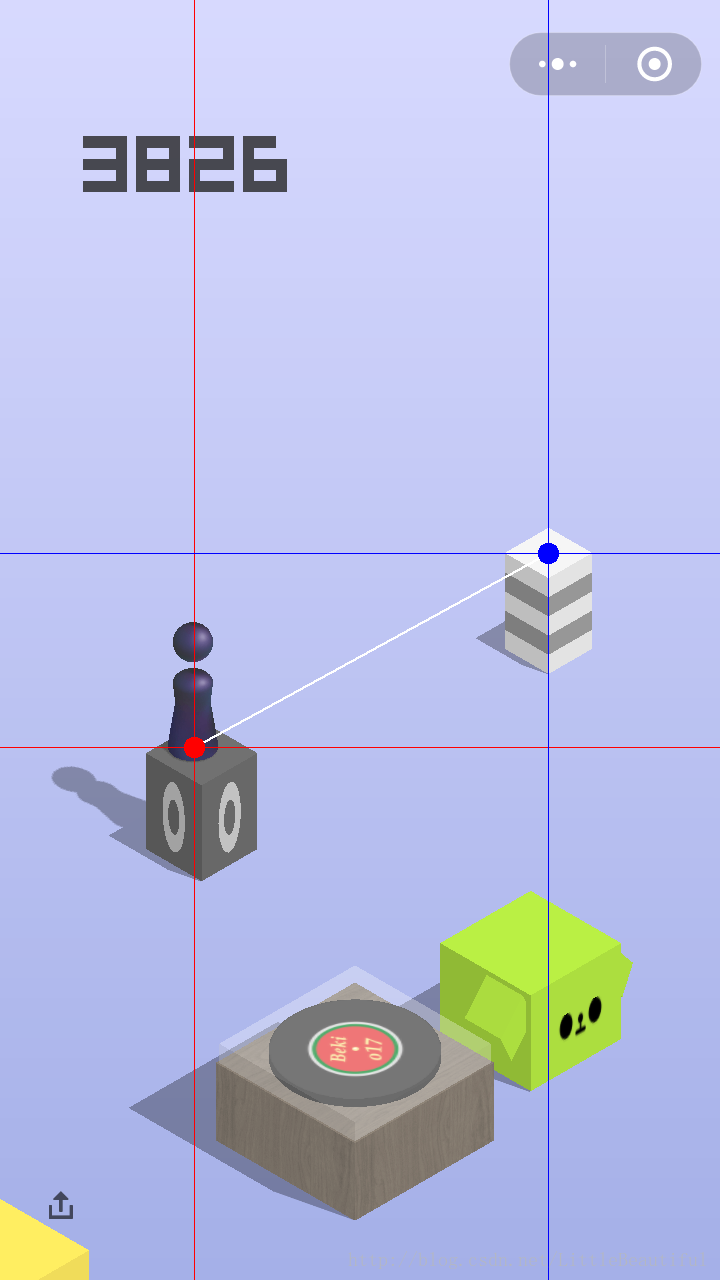
**（2）、测试求得加分的中间值**



694~732之间都可以拿到加分，此时取中间值(694+732)/2=713

**（3）、duration值的计算：**

假设我们截屏的效果是如下：



从图中可以看到，时间的值跟开始位置到结束位置的距离有关。

假设时间是t，距离是s。公式应该是s = at

基本思路：两点之间的距离乘以一个时间系数。

所以要从截图上识别出起跳位置的坐标(x1,y1)和目标位置的坐标(x2,y2)。

起跳位置的坐标：小人的底座中心点

目标位置的坐标：目标菱形的中心点

然后计算这两点之间的距离（欧氏距离）：sqrt((x1-x2)2+(y1-y2)2)

# 6、截屏代码

**（1）、创建项目，新建 img目录，把所有截屏的图片都放到该目录下（原则上每跳一步都需要截屏一次）**

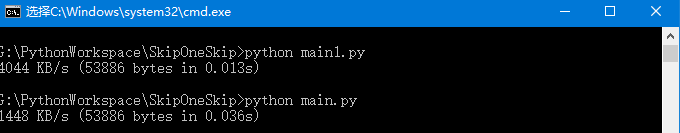
operation.py

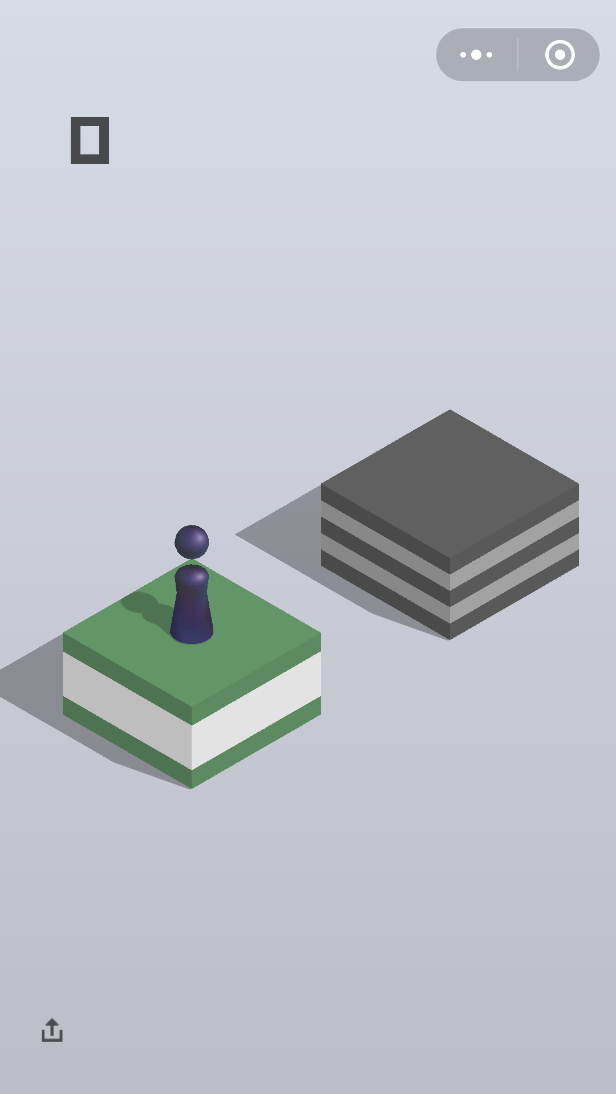
|  |
| --- |
| **import** os **import** datetime *# 实现控制Android设备等相关的操作* **class** Operation:  *# 构造方法* **def** \_\_init\_\_(self):  **pass** *# 截屏* **def** screen\_cap(self):  filename = time = datetime.datetime.now().strftime(**"%H%M%S"**) + **".png"** *# 截屏并保存到手机的目录上* cmd = **"adb shell screencap -p /sdcard/"** + filename  os.system(cmd)  *# 把手机目录上的文件拷贝到PC上* cmd = **"adb pull /sdcard/"** + filename + **" img/"** + filename  os.system(cmd) |

main.py

|  |
| --- |
| **from** operation **import** \*  **def** test\_screen\_cap():  op = Operation()  op.screen\_cap()  *# 测试截屏* test\_screen\_cap() |

**（2）、截屏代码运行结果**





# 7、显示图片的代码

**（1）、安装matplotlib库**

|  |
| --- |
| pip install matplotlib |

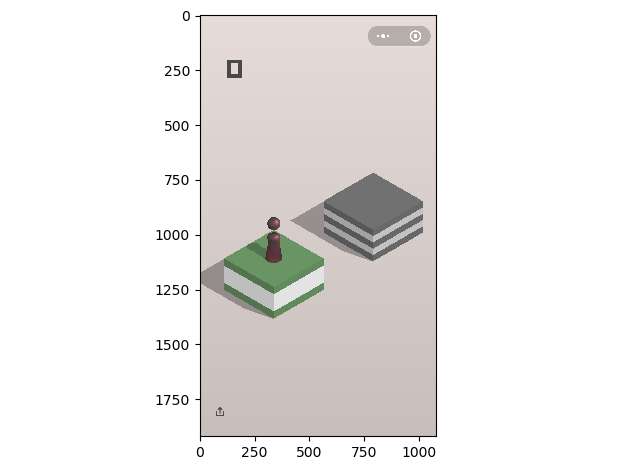
**（2）、图片显示代码**

draw.py

|  |
| --- |
| **import** matplotlib.pyplot **as** plt *# 绘图* **import** cv2 *# 读取图片文件  # 实现显示图片 绘制图片等功能* **class** Draw:  *# 构造器* **def** \_\_init\_\_(self):  *# 初始化图像plt对象* self.fig = plt.figure()   *# 显示图片* **def** show\_pic(self, filename,scale=1):  *# 读取图像* img = cv2.imread(filename)  *# 调整显示的比例* img = cv2.resize(img, (0,0), fx=scale, fy=scale)  *# 显示图像* plt.imshow(img)  plt.show() |

main.py

|  |
| --- |
| *# 测试显示图片* **def** test\_show\_pic():  draw = Draw()  draw.show\_pic(**"img/143041.png"**) |



# 8、计算两点之间的欧氏距离

创建文件algorithm.py文件，添加计算欧氏距离的方法：

algorithm.py

|  |
| --- |
| *# 算法类* **class** Algorithm:  *# 构造器* **def** \_\_init\_\_(self):  **pass** *# 计算两点之间的欧氏距离  # p1和p2表示两个点 用元组来表示* **def** euclidean\_distance(self,p1,p2):  **return** ((p1[0] - p2[0])\*\*2 + (p1[1] - p2[1])\*\*2) \*\* 0.5 |

main.py

|  |
| --- |
| *# 测试计算欧氏距离* **def** test\_euclidean\_distane():  algorithm = Algorithm()  p1 = (3,4)  p2 = (6,8)  d = algorithm.euclidean\_distance(p1,p2)  **print**(d) |



# 9、寻找关键坐标——框架

查找起跳点的坐标和目标点的坐标：

在algorithm.py中添加方法find\_point()

|  |
| --- |
| *# 寻找关键坐标 # 返回值1,2 start\_x, start\_y 起跳点的坐标 170,555 # 返回值3,4 end\_x, end\_y 目标点的坐标 395,425* **def** find\_point(self):  start\_x = start\_y = 0  end\_x = end\_y = 0  **return** start\_x, start\_y, end\_x, end\_y |

main.py

|  |
| --- |
| *# 测试寻找关键坐标* **def** test\_find\_point():  algorithm = Algorithm()  start\_x, start\_y, end\_x, end\_y = algorithm.find\_point()  **print**(**"start\_point:"**, start\_x, start\_y)  **print**(**"end\_point:"**, end\_x, end\_y) |



# 10、获取每一个点的RGB值

|  |
| --- |
| # 寻找关键坐标  # 返回值1,2 piece\_x, piece\_y 起跳点的坐标 170,555  # 返回值3,4 board\_x, board\_y 目标点的坐标 395,425  def **find\_point**(*self*,im):  piece\_x = piece\_y = 0  board\_x = board\_y = 0    # 图像的大小  w,h = im.size # (540, 960)  # 加载图像  im\_pixel = im.load()    # 遍历图像中的每一个点  # 遍历每一行  for i in range(h):  # 遍历每一列  for j in range(w):    pixel = im\_pixel[j,i]  print(*"i = "*, i, *",j = "*, j, *"pixel = "*, pixel) |

测试代码：

|  |
| --- |
| # 测试寻找关键坐标  def **test\_find\_point**():  op = Operation()  im = op.screen\_cap()  algorithm = Algorithm()  start\_x, start\_y, end\_x, end\_y = algorithm.find\_point(im)  print(*"start\_point:"*, start\_x, start\_y)  print(*"end\_point:"*, end\_x, end\_y) |

# 11、寻找关键坐标——起跳坐标

算法策略：获取小人的底座中心点的值作为起跳点。

（1）获取小人的所有像素点中y坐标的最大值

（2）在小人y坐标的最大值那些像素点中，计算出x的平均值，作为小人底座的x的值。

（3）y坐标的最大值减去一个偏移值，就作为小人底座的y值。（注意：该偏移值不同的设备是不同的，同一台设备不同场景下是一样的）

比如教师机的设备中最低点的值是(168,565)，中心值是 (168,555)，从而计算出偏移值为565-555=10



## 11.1 获取小人y坐标的最大值

需要从上往下一行行扫描像素点，直到找到小人位置：

|  |
| --- |
| # 寻找关键坐标  # 返回值1,2 piece\_x, piece\_y 起跳点的坐标 170,555  # 返回值3,4 board\_x, board\_y 目标点的坐标 395,425  def **find\_point**(*self*,im):  piece\_x = piece\_y = 0  board\_x = board\_y = 0    # 图像的大小  w,h = im.size # (540, 960)  # 加载图像  im\_pixel = im.load()    # 记录y的最大值  piece\_y\_max = 0    # 1 计算出起跳点 就是小人底座的中心点  # 1.1 获取小人的所有像素点中y坐标的最大值  # 遍历图像中的每一个点  # 遍历每一行  for i in range(h):  # 遍历每一列  for j in range(w):    pixel = im\_pixel[j,i]  #print("i = ", i, ",j = ", j, "pixel = ", pixel)    # 判断pixel是否小人所在的位置  # 当该点的RGB值约为55,59,102的时候就可以认为是小人所在的像素点了  if(50 < pixel[0] < 60 and 53 < pixel[1] < 63 and 95 < pixel[2] < 110):  # 记录下y的值  if i > piece\_y\_max:  piece\_y\_max = i    print(*"piece\_y\_max = %d"* % (piece\_y\_max,))    # 1.2 在小人y坐标的最大值那些像素点中，计算出x的平均值，作为小人底座的x的值。    # 1.3 y坐标的最大值减去一个偏移值，就作为小人底座的y值。（注意：该偏移值不同的设备是不同的，同一台设备不同场景下是一样的）    return piece\_x, piece\_y, board\_x, board\_y |

## 11.2 获取小人底座的x坐标和y坐标

记录下小人所有的点：

|  |
| --- |
| # 寻找关键坐标  # 返回值1,2 piece\_x, piece\_y 起跳点的坐标 170,555  # 返回值3,4 board\_x, board\_y 目标点的坐标 395,425  def **find\_point**(*self*,im):  piece\_x = piece\_y = 0  board\_x = board\_y = 0    # 图像的大小  w,h = im.size # (540, 960)  # 加载图像  im\_pixel = im.load()  # 记录小人所有的点  points = []    # 记录y的最大值  piece\_y\_max = 0    # 1 计算出起跳点 就是小人底座的中心点  # 1.1 获取小人的所有像素点中y坐标的最大值  # 遍历图像中的每一个点  # 遍历每一行  for i in range(h):  # 遍历每一列  for j in range(w):    pixel = im\_pixel[j,i]  #print("i = ", i, ",j = ", j, "pixel = ", pixel)    # 判断pixel是否小人所在的位置  # 当该点的RGB值约为55,59,102的时候就可以认为是小人所在的像素点了  if(50 < pixel[0] < 60 and 53 < pixel[1] < 63 and 95 < pixel[2] < 110):  # 把当前的点添加到points数组中  points.append((j,i)) # (x,y)  # 记录下y的值  if i > piece\_y\_max:  piece\_y\_max = i    print(*"piece\_y\_max = %d"* % (piece\_y\_max,))  # 1.2 在小人y坐标的最大值那些像素点中，计算出x的平均值，作为小人底座的x的值。  bottom\_x = []  for x,y in points:  if y == piece\_y\_max:  bottom\_x.append(x)    piece\_x = sum(bottom\_x) // len(bottom\_x)  print(*"piece\_x = %d"* % (piece\_x,))    # 1.3 y坐标的最大值减去一个偏移值，就作为小人底座的y值。（注意：该偏移值不同的设备是不同的，同一台设备不同场景下是一样的）  piece\_y = piece\_y\_max - *self*.piece\_base\_height  print(*"piece\_y = %d"* % (piece\_y,))  return piece\_x, piece\_y, board\_x, board\_y |

# 12、优化程序

无论是起跳位置，还有目标位置。都只取垂直的中间的1/3样式进行扫描。

|  |
| --- |
| # 1 计算出起跳点 就是小人底座的中心点  # 1.1 获取小人的所有像素点中y坐标的最大值  # 遍历图像中的每一个点  # 遍历每一行  for i in range(h / 3, h \* 2 / 3): |

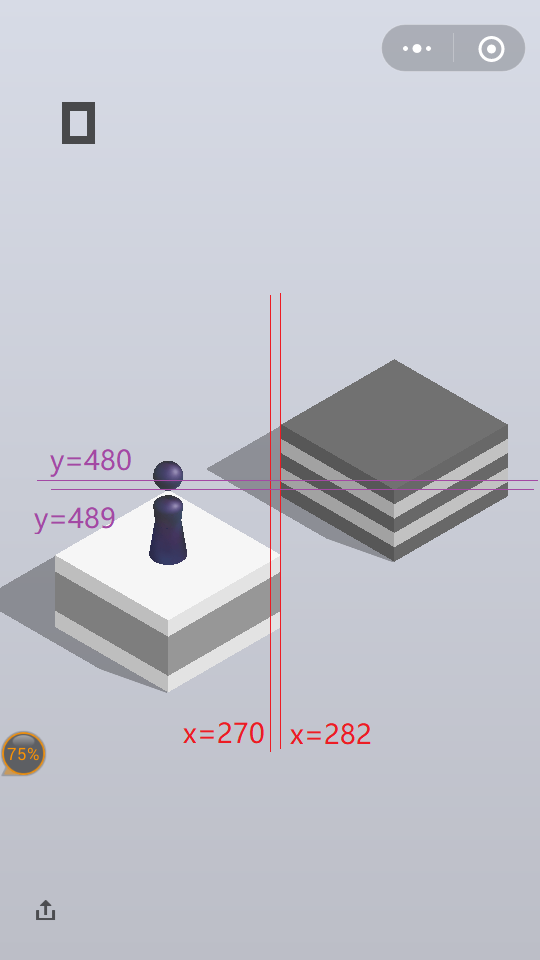
# 13、寻找关键坐标——目标坐标

## 13.1 获取目标坐标的x值

|  |
| --- |
| # 寻找关键坐标  # 返回值1,2 piece\_x, piece\_y 起跳点的坐标 170,555  # 返回值3,4 board\_x, board\_y 目标点的坐标 395,425  def **find\_point**(*self*,im):  piece\_x = piece\_y = 0  board\_x = board\_y = 0    # 图像的大小  w,h = im.size # (540, 960)  # 加载图像  im\_pixel = im.load()  # 记录小人所有的点  points = []    # 记录y的最大值  piece\_y\_max = 0    # 1 计算出起跳点 就是小人底座的中心点  # 1.1 获取小人的所有像素点中y坐标的最大值  # 遍历图像中的每一个点  # 遍历每一行  for i in range(h // 3, h \* 2 // 3):  # 遍历每一列  for j in range(w):    pixel = im\_pixel[j,i]  #print("i = ", i, ",j = ", j, "pixel = ", pixel)    # 判断pixel是否小人所在的位置  # 当该点的RGB值约为55,59,102的时候就可以认为是小人所在的像素点了  if(50 < pixel[0] < 60 and 53 < pixel[1] < 63 and 95 < pixel[2] < 110):  # 把当前的点添加到points数组中  points.append((j,i)) # (x,y)  # 记录下y的值  if i > piece\_y\_max:  piece\_y\_max = i    print(*"piece\_y\_max = %d"* % (piece\_y\_max,))  # 1.2 在小人y坐标的最大值那些像素点中，计算出x的平均值，作为小人底座的x的值。  bottom\_x = []  for x,y in points:  if y == piece\_y\_max:  bottom\_x.append(x)    piece\_x = sum(bottom\_x) // len(bottom\_x)  print(*"piece\_x = %d"* % (piece\_x,))    # 1.3 y坐标的最大值减去一个偏移值，就作为小人底座的y值。（注意：该偏移值不同的设备是不同的，同一台设备不同场景下是一样的）  piece\_y = piece\_y\_max - *self*.piece\_base\_height  print(*"piece\_y = %d"* % (piece\_y,))  points = []  # 只取中间1/3进行扫描  for i in range(h // 3, h \* 2 // 3):  if len(points) > 0:  break  # 取坐标的一个点作为背景的参照物  last\_pixel = im\_pixel[0,i]  # 逐个扫描右边的点  for j in range(w):  pixel = im\_pixel[j,i]  # 把当前点与最左边的点比较 如果RGB差异比较大 则认为是目标点  if(abs(pixel[0] - last\_pixel[0])  + abs(pixel[1] - last\_pixel[1])  + abs(pixel[2] - last\_pixel[2]) > 10):  points.append((j,i))  top\_x = []  for x,y in points:  top\_x.append(x)  board\_x = sum(top\_x) // len(top\_x)  print(*"board\_x = %d"* % (board\_x,))  return piece\_x, piece\_y, board\_x, board\_y |

## 13.2 获取目标坐标的y值

取屏幕宽和高的一半（x=270和y=480）



我们会发现，目标格子的边沿（x=282，y=489）和这个是差不多的(y的偏差是9，x的偏差是12)

以后每次跳动的时候，假如已经知道目标格子的边沿，和目标坐标的x值，就可以很轻松计算出目标坐标的y值。

注意：每个格子的宽和高的比例是相同的。

左：（281,424）

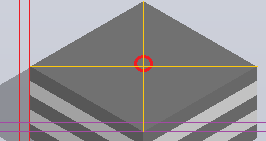
右：（501,424）

上：（394,360）

下：（394,489）

中：（394,424）

高和宽的比例：(489-360)/(501-281) =129/220。假设该值为p



已经知道目标坐标的x值，求目标坐标的y值：

|  |
| --- |
| # 寻找关键坐标  # 返回值1,2 piece\_x, piece\_y 起跳点的坐标 170,555  # 返回值3,4 board\_x, board\_y 目标点的坐标 395,425  def **find\_point**(*self*,im):  piece\_x = piece\_y = 0  board\_x = board\_y = 0    # 图像的大小  w,h = im.size # (540, 960)  # 加载图像  im\_pixel = im.load()  # 记录小人所有的点  points = []    # 记录y的最大值  piece\_y\_max = 0    # 1 计算出起跳点 就是小人底座的中心点  # 1.1 获取小人的所有像素点中y坐标的最大值  # 遍历图像中的每一个点  # 遍历每一行  for i in range(h // 3, h \* 2 // 3):  # 遍历每一列  for j in range(w):    pixel = im\_pixel[j,i]  #print("i = ", i, ",j = ", j, "pixel = ", pixel)    # 判断pixel是否小人所在的位置  # 当该点的RGB值约为55,59,102的时候就可以认为是小人所在的像素点了  if(50 < pixel[0] < 60 and 53 < pixel[1] < 63 and 95 < pixel[2] < 110):  # 把当前的点添加到points数组中  points.append((j,i)) # (x,y)  # 记录下y的值  if i > piece\_y\_max:  piece\_y\_max = i    print(*"piece\_y\_max = %d"* % (piece\_y\_max,))  # 1.2 在小人y坐标的最大值那些像素点中，计算出x的平均值，作为小人底座的x的值。  bottom\_x = []  for x,y in points:  if y == piece\_y\_max:  bottom\_x.append(x)    piece\_x = sum(bottom\_x) // len(bottom\_x)  print(*"piece\_x = %d"* % (piece\_x,))    # 1.3 y坐标的最大值减去一个偏移值，就作为小人底座的y值。（注意：该偏移值不同的设备是不同的，同一台设备不同场景下是一样的）  piece\_y = piece\_y\_max - *self*.piece\_base\_height  print(*"piece\_y = %d"* % (piece\_y,))  # 2计算 目标格子的中心点  # 2.1计算目标格子的x值  points = []  # 只取中间1/3进行扫描  for i in range(h // 3, h \* 2 // 3):  if len(points) > 0:  break  # 取坐标的一个点作为背景的参照物  last\_pixel = im\_pixel[0,i]  # 逐个扫描右边的点  for j in range(w):  pixel = im\_pixel[j,i]  # 把当前点与最左边的点比较 如果RGB差异比较大 则认为是目标点  if(abs(pixel[0] - last\_pixel[0])  + abs(pixel[1] - last\_pixel[1])  + abs(pixel[2] - last\_pixel[2]) > 10):  points.append((j,i))    top\_x = []  for x,y in points:  top\_x.append(x)  board\_x = sum(top\_x) // len(top\_x)  print(*"board\_x = %d"* % (board\_x,))    # 2.2计算目标格式子y值  # 屏幕中心的值  center\_x = w / 2 + 12 # x的偏差是12  center\_y = h / 2 + 9 # y的偏差是9  # 格子高和宽的比例  height\_per\_width = 129 / 220  # 计算出目标格子的y值(需要转换成整数)  board\_y = int(center\_y - height\_per\_width \* (board\_x - center\_x))  print(*"board\_y = %d"* % (board\_y,))  return piece\_x, piece\_y, board\_x, board\_y |

## 13.3 区分从左往右跳和从右往左跳

|  |
| --- |
| # 寻找关键坐标  # 返回值1,2 piece\_x, piece\_y 起跳点的坐标 170,555  # 返回值3,4 board\_x, board\_y 目标点的坐标 395,425  def **find\_point**(*self*,im):  …………    # 2.2计算目标格式子y值  # 屏幕中心的值  center\_x = w / 2 + 12 # x的偏差是12  center\_y = h / 2 + 9 # y的偏差是9  # 格子高和宽的比例  height\_per\_width = 129 / 220  # 计算出目标格子的y值(需要转换成整数)  # 从piece\_x调到board\_x 如果piece\_x < board\_x则表示从左往右跳  # 如果piece\_x > board\_x 则表示从右往左跳  if piece\_x < board\_x:  board\_y = int(center\_y - height\_per\_width \* (board\_x - center\_x))  else: # 从右往左跳  board\_y = int(center\_y + height\_per\_width \* (board\_x - center\_x))  print(*"board\_y = %d"* % (board\_y,))  return piece\_x, piece\_y, board\_x, board\_y |

## 13.4 计算两点间的距离

在测试代码中，通过返回的piece\_x、piece\_y、board\_x、board\_y就可以计算出两点之间的欧氏距离了

main.py

|  |
| --- |
| # 测试寻找关键坐标  def **test\_find\_point**():  op = Operation()  im = op.screen\_cap()  algorithm = Algorithm()  start\_x, start\_y, end\_x, end\_y = algorithm.find\_point(im)  print(*"start\_point:"*, start\_x, start\_y)  print(*"end\_point:"*, end\_x, end\_y)    start\_point = (start\_x, start\_y)  end\_point = (end\_x, end\_y)  distance = algorithm.euclidean\_distance(start\_point, end\_point)  print(*"distance = %f"* % distance) |

# 14、初步估算距离与时间的比例

algorithm.py

|  |
| --- |
| # 距离与时间的转换  def **distance\_to\_time**(*self*, distance):  # 当0分的时候 距离为 261.222128 时间为730  p = 730 / 261.222128 # 该算法后面待优化  press\_time = distance \* p  return press\_time |

# 15、控制屏幕进行跳动

operation.py

|  |
| --- |
| # 控制屏幕进行跳动  def **jump**(*self*, src, dst, press\_time):  press\_time = int(press\_time)  cmd = *"adb shell input swipe %d %d %d %d %d"* % (  int(src[0]), int(src[1]),  int(dst[0]), int(dst[1]),  press\_time  )  print(cmd)  os.system(cmd) |

main.py

|  |
| --- |
| def **test\_jump**():  algorithm = Algorithm()  op = Operation()  im = op.screen\_cap()  start\_x, start\_y, end\_x, end\_y = algorithm.find\_point(im)  start\_point = (start\_x, start\_y)  end\_point = (end\_x, end\_y)  distance = algorithm.euclidean\_distance(start\_point, end\_point)  press\_time = algorithm.distance\_to\_time(distance)  op.jump(start\_point, end\_point, press\_time) |

# END