Human-computer interaction
Ludo game
based on image recognition

TEAM : Hello full mark !
Guo Yi, Cheng Bo Yuan
Ni Jun Yan, Reina Honda

Project Report

June 9, 2019

1 Basic Information

Project title: Human-computer interaction Ludo game based on image recognition
Team name: Hello full mark
Team members: Cheng Boyuan, Ni Junyan, Bentian Linai, Guo Yi

2 Problem Introduction

2.1 The background of Ludo games

Ludo is a highly simplistic version of Pachisi, a game that originated in the 6th century in India. This game is played by younger children all over the country. In this board game 2 to 4, players race their tokens from start to finish according to the dice rolls. Various variations are seen in the way people play Ludo.

Ludo originated in India by the 6th century. The earliest evidence of this game in India is the depiction of boards on the caves of Ajanta. This game was played by the Mughal emperors of India; a notable example being that of Jalaluddin Muhammad Akbar.

Variations of the game made it to England during the late 19th century. One which appeared around 1896 under the name of Ludo was then successfully patented. Special areas of the Ludo board are typically coloured bright yellow, green, red, and blue. Each player is assigned a colour and has four tokens of matching colour (originally bone discs but today often made of cardboard or plastic). The board is normally square with a cross-shaped game track, with each arm of the cross consisting of three columns of squaresusually six squares per column. The middle columns usually have five squares coloured, and these represent a player's home column. A sixth coloured square not on the home column is a player's starting square. At the centre of the board is a large finishing square often composed of triangles in the four colours atop the players' home columns C thus forming "arrows" pointing to the finish.

2.2 Project innovation

We made appropriate improvements to the game based on traditional Ludo games. First of all, in the game to add props features, such as bombs and stars, so that the game's variability is more abundant, the emergence of a large number of props in the late time of the game can also make the difficulty of the game further.At the same time, it also adds to the strategic nature of the game. Second, increase the composition of human-computer interaction, when the player touches the bomb, he needs to perform a "video form" of rock-paper-scissors with the computer. If he loses, he will return to the starting point and start again, adding fun to the game.

2.3 The libraries we use

2.3.1 Pygame

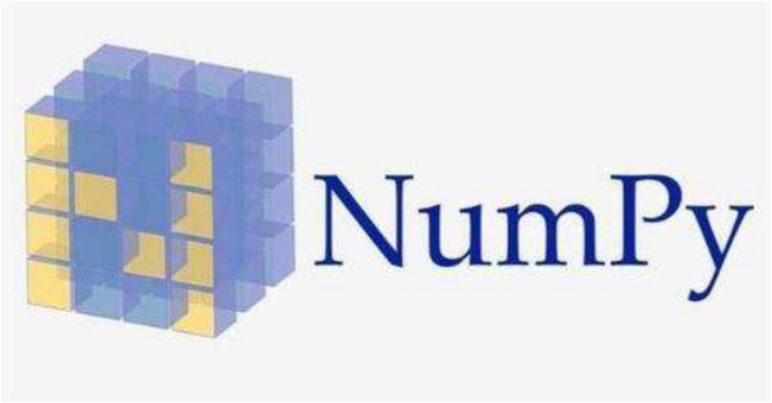
Pygame is a Python wrapper module for the SDL multimedia library. It contains python functions and classes that will allow you to use SDLs support for playing cdroms, audio and video output, and keyboard, mouse and joystick input. We use pygame for the Ludo game part.

2.3.2 OpenCv2

Open Source Computer Vision Library was released under a BSD license and hence its free for both academic and commercial use. OpenCV2 was designed for computational efficiency and with a strong focus in real-time applications. In this project, the rock-paper-scissors part is created mainly using Opencv.

2.3.3 numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. In the whole project, we use the numpy library for scientific computing.



3 The rules of the game

3.1 Equipment

A Ludo board is is square with a pattern on it in the shape of a cross. The middle squares form the home column for each colour and cannot be landed upon by other colours. The middle of the cross forms a large square which is the 'home' area and which is divided into 4 home triangles, one of each colour. At each corner, separate to the main circuit are coloured squares where the pieces are placed to begin.

Counters start their circuit one square in from the end of the arm and adjacent to the starting square. Avoid modern boards which incorrectly place the first square at the end of the arm.?Each player chooses one of the 4 colours (green, yellow, red or blue) and places the 4 pieces of that colour in the corresponding starting square. And there will be bombs and stars in the game. If the die is 6, a bomb can be placed on the board. If the die is 1, a star can be placed on the board.

3.2 Play

A player must throw a 6 or 1 to move a piece from the starting circle onto the first square on the track.

Each throw, the player decides which piece to move. If no piece can legally move according to the number thrown, play passes to the next player.

If the player reaches a position with a bomb, the game of rockpaperscissors will be played. The loser needs to return the piece to its starting point.

If the player reaches a position with a star, the pieces will go directly to the position of six squares away from the end point.

3.3 Winning

When all the pieces of the player reach the end point.

4 The process of programming

4.1 Basic attributes

Board size = 1000*700
Padding = 10
Outline Width = 5
title = 'Ludo Board'
box size= 50*50

4.2 Recognition of gesture

4.2.1 Prepare the library

```
camera = cv2.VideoCapture(0)

camera.set(10, 200)

cv2.namedWindow('trackbar')

cv2.createTrackbar('trh1', 'trackbar', threshold, 100, printThreshold)
```

4.3 Capture and convert image

```
while camera.isOpened():  # capture and convert image

    ret, frame = camera.read()

    threshold = cv2.getTrackbarPos('trh1', 'trackbar')

    frame = cv2.bilateralFilter(frame, 5, 50, 100)  # smoothing filter

    frame = cv2.flip(frame, 1)  # flip the frame horizontally

    cv2.rectangle(frame, (int(cap_region_x_begin * frame.shape[1]), 0),

                  (frame.shape[1], int(cap_region_y_end * frame.shape[0])), (255, 0, 0), 2)

    cv2.imshow('original', frame)
```

4.4 Convert the image into binary image

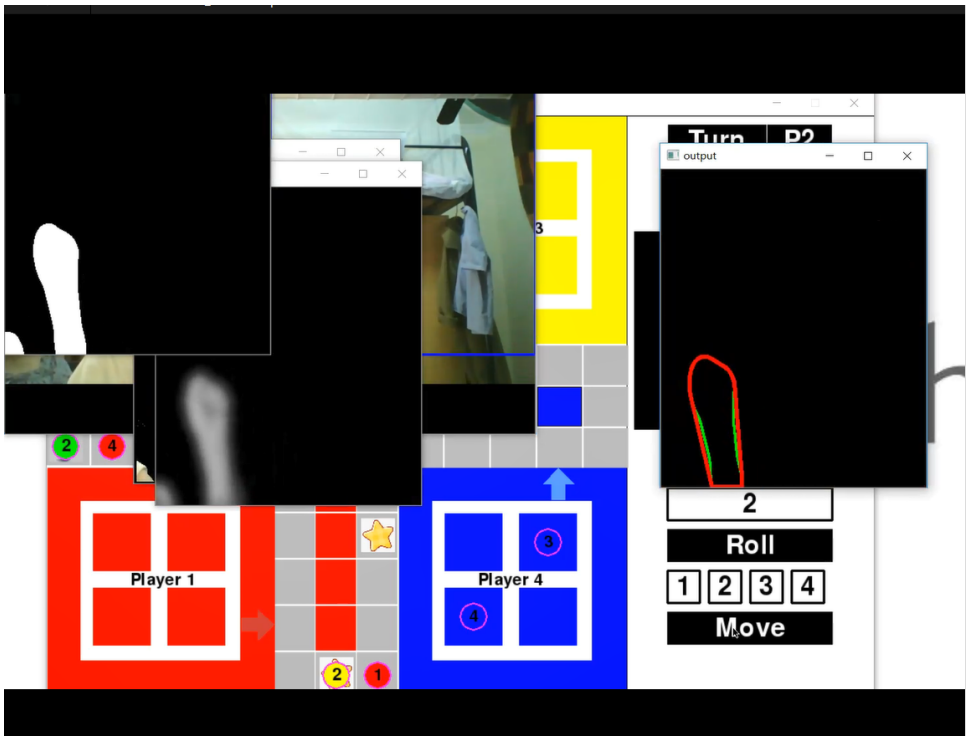
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

blur = cv2.GaussianBlur(gray, (blurValue, blurValue), 0)

cv2.imshow('blur', blur)

ret, thresh = cv2.threshold(blur, threshold, 255, cv2.THRESH_BINARY)

cv2.imshow('ori', thresh)
```

4.5 Get the coutour

```
Flag = True
if len(hull) > 3:

    defects = cv2.convexityDefects(res, hull)   # finding defects

    if type(defects) != type(None):  # avoid crashing.   (BUG not found)

        cnt = 0

        for i in range(defects.shape[0]):  # calculate the angle

            s, e, f, d = defects[i][0]

            start = tuple(res[s][0])

            end = tuple(res[e][0])

            far = tuple(res[f][0])

            a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)

            b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)

            c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)

            angle = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c))  # cosine theorem

            if angle <= math.pi / 2:  # angle less than 90 degree, treat as fingers

                cnt += 1

                cv2.circle(drawing, far, 8, [211, 84, 0], -1)
```

In this part, we capture and convert image into binary image. Then we compare the extracted feature points with the gesture dictionary and then determine the gesture and shape. Find the largest contour according to the contrast with the background.

Finally, according to the coordinates of concave and convex points in the image, the angle between two fingers is calculated by using the cosine theorem. Fingers can be identified by calculating the angle less than 90 degrees.



5 Create the main game

5.1 Draw the game interface

```
# 绘制摇骰子的按钮
def showButtonRoll():
    pygame.draw.rect(dispsurf,BLACK,(750,500,200,40))
    renderText("Roll",850,520,32,WHITE)

# 绘制移动的按钮
def showButtonMove():
    pygame.draw.rect(dispsurf,BLACK,(750,600,200,40))
    renderText("Move",850,620,32,WHITE)

# 绘制骰子所在的方框
def showDiceBox(n):
    pygame.draw.rect(dispsurf,BLACK,(750,450,200,40),3)
    renderText(str(n),850,470,32,BLACK)

# 绘制当前行动玩家的方框
def showTurnBox(n):
    pygame.draw.rect(dispsurf,BLACK,(750,10,200,40))
    pygame.draw.rect(dispsurf,WHITE,(750,10,200,40),1)
    pygame.draw.line(dispsurf,WHITE,(870,10),(870,50),1)
    renderText("Turn",810,30,32,WHITE)
    renderText("P"+str(n),910,30,32,WHITE)
```

We first design all the tools needed for the Ludo game. This includes a button to press when rolling dice, a button to move after rolling dice and a box to remind players about whose turn it is.

Then we draw the game board to show the position of the tokens. The major element of a game board is considered to be the colored track, whose color shows the start place and ending place of a particular player.

The drawing process is not complicated, mainly to calculate the coordinates of each

```
    # 绘制棋子初始位置的边框
    def showLudoButtons():
        pygame.draw.rect(dispsurf,BLACK,(750,550,40,40),3)
        renderText(str(1),770,570,32,BLACK)
    # 绘制棋盘
    def drawBoard():
        # 绘制外边框
        pygame.draw.line(dispsurf,BLACK,(0,0),(1000,0))
        pygame.draw.line(dispsurf,BLACK,(701,0),(701,700))
        # 绘制终点区域
        C = (350,350)
        BL = (275,425)
        BR = (425,425)
        pygame.draw.polygon(dispsurf,RED,(C,BR,BL),0)
```

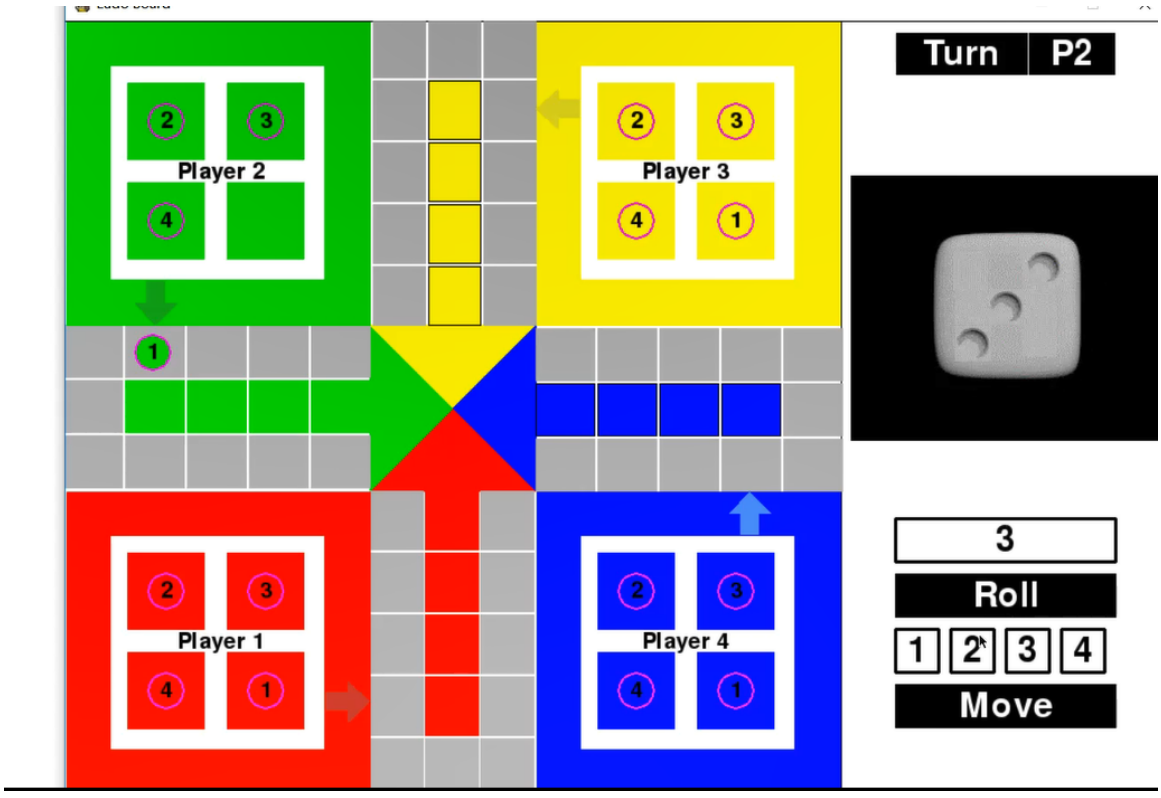
```
    # 绘制上下左右四个轨道部分
    # 下方轨道
    list1 = [(275, 425,GREY), (275, 481,GREY), (275, 537,GREY), (275, 593,GREY), (275, 649,GREY),
(325, 425,RED), (325, 481,RED), (325, 537,RED), (325, 593,RED), (325, 649,GREY),
(375, 425,GREY), (375, 481,GREY), (375, 537,GREY), (375, 593,GREY), (375, 649,GREY)]
    for item in list1:
        if(item[2] == RED):
            pygame.draw.rect(dispsurf,item[2],(item[0],item[1],49,54),0)
        elif(item[2] == GREY):
            pygame.draw.rect(dispsurf,item[2],(item[0],item[1],49,54),0)
        else:
            pygame.draw.rect(dispsurf,BLACK,(item[0],item[1],50,54),1)

    # 绘制起点轨道
    colors = [RED,GREEN,YELLOW,BLUE]
    p_home = [(0,425),(0,0),(425,0),(425,425)]

    for i in range(1,5):
        pygame.draw.rect(dispsurf,colors[i-1],(p_home[i-1][0]+1,p_home[i-1][1]+1,275,275))
        pygame.draw.rect(dispsurf,WHITE,(p_home[i-1][0]+41,p_home[i-1][1]+41,193,193))
        xx = p_home[i-1][0]+41
        yy = p_home[i-1][1]+41
        pygame.draw.rect(dispsurf,colors[i-1],(xx+15,yy+15,70,70))
        pygame.draw.rect(dispsurf,colors[i-1],(xx+105,yy+15,70,70))
        pygame.draw.rect(dispsurf,colors[i-1],(xx+15,yy+105,70,70))
        pygame.draw.rect(dispsurf,colors[i-1],(xx+105,yy+105,70,70))
    # 绘制起点箭头
    # 左上方
    pygame.draw.rect(dispsurf,(50,200,50),(73,235,17,20))
    pygame.draw.polygon(dispsurf,(50,200,50),((63,255),(100,255),(81.5,275)),0)
    # 绘制区域标号
    renderText("Player 2",140,137,20,BLACK)
```

point, and to match the corresponding color for each part. In addition, some images need to be inserted into the text, for which we define the Rendertext function for text insertion. A big difficulty in this part of the production process is that the coordinates of each point are difficult to calculate accurately. Another difficulty lies in the drawing of irregular figures.

Of course, one of the drawbacks of the game is that there is no distinction between the colors of the individual grids (if a distinction is needed, a grid class should be defined separately to determine whether it is flying to its corresponding position), and the corners of the road sections are not connected. This is also the direction that needs improvement in the future.

Now the game interface is finished as shown below.

```
'''
 四个玩家棋子的初始位置列表
'''
position1 = [(91,516), (181,516), (91,606), (181,606)]
position2 = [(91,91), (181,91), (91,181), (181,181)]
position3 = [(516,91), (606,91), (516,181), (606,181)]
position4 = [(516,516), (606,516), (516,606), (606,606)]


'''
 四个玩家初始的棋子状态1列表
'''
initiated1 = [(0,0),(0,0),(0,0),(0,0)]
initiated2 = [(0,0),(0,0),(0,0),(0,0)]
initiated3 = [(0,0),(0,0),(0,0),(0,0)]
initiated4 = [(0,0),(0,0),(0,0),(0,0)]

while True:
    dispsurf.fill(WHITE)     # 背景颜色设置为白色
    drawBoard()              # 绘制画布
    showButtonRoll()         # 绘制摇骰子按钮
    showButtonMove()         # 绘制移动按钮
    dispdiceroll(n)          # 显示骰子的状态

    if(mc == 0):             # 当摇骰子的时候保持之前所显示的数字
        showDiceBox(0)
    else:
        showDiceBox(n)       # 摇完骰子后，显示摇到的数字

    showLudoButtons()        # 显示用于控制第几个棋子行走的按钮
    showTurnBox(p_id)        # 显示当前正在激活的玩家
```



5.2 Define class

Here we get the game board and we need tokens to be put on it. We define a class named player to store and define their behaviors while playing. The player class contains many functions, such as recording the color of the piece, recording the track of the piece, judging whether the piece leaves the starting point, judging whether the piece reaches the end point, moving the piece forward, returning the piece to the starting point, and playing the piece. Leap to the finish, show the pieces, and more.See the specific introduction of these functions in the coding line.

```
# 玩家类
class Player (LudoButton):
    score = 0    # 记录每个玩家的得分
    def __init__(self,color,track):
        self.color = color    # 该玩家棋子的颜色
        self.LudobuttonList = [LudoButton() for i in range(1,5)]   # 初始化四个棋子
        self.finalstatus = [0,0,0,0]                               # 棋子是否到达终点的标记
        self.initialstatus = [0,0,0,0]                             # 棋子是否离开待行区的标记
        self.track = track                                         # 玩家运动的轨道
        self.currentposition = [-1,-1,-1,-1] # 当前每个棋子的位置（用track列表的下标表示，-1代表在准备区

    def placeLudoButtons(self,position,initiated):
        '''
        将棋子放在各自的位置
        position棋子应当摆放的位置
        initiated表示棋子的初始位置
        '''
        for i in range(4):
            if(initiated[i] == position[i]):
                kkk=32
            else:
                pygame.draw.circle(dispsurf,PURPLE,position[i-1],17,0)
                self.drawLudoButton(self.color,position[i-1][0],position[i-1][1],i+1)

    def makeAMove(self,current):
        '''
        对棋子进行移动一步
        current代表这个棋子移动前的位置
        '''
        pygame.draw.circle(dispsurf,self.color,self.track[current + 1],self.radius,0)

    def backToStart(self):
        '''
        对棋子进行移动一步
        current代表这个棋子移动前的位置
        '''
        pygame.draw.circle(dispsurf,self.color,self.track[0],self.radius,0)

    def backToEnd(self):
        '''
        将棋子移动到终点轨道
        '''
        pygame.draw.circle(dispsurf,self.color,self.track[41],self.radius,0)
```

Similarly, we define a class named Ludo Button in Python, adding certain attributes to it. Circle is drawn to represent the token for our Ludo game.

```
# 棋子类
class LudoButton:
    radius = 15
    center_x = 0
    center_y = 0

    #绘制一个棋子
    def drawLudoButton(self,color,center_x,center_y,number):
        self.center_x = center_x
        self.center_y = center_y
        pygame.draw.circle(dispsurf,color,(self.center_x,self.center_y),self.radius,0)
        renderText(str(number), self.center_x,self.center_y, 20, BLACK)
        ##这个地方加上棋子的编号
```

5.3 Realize dice-rolling

```
# 显示骰子投掷的过程
def diceroll(n,c,b,tick,In):
    index = 1;
    location = (710, 140)  #骰子投掷的位置坐标
    #开始展示投掷骰子的动画
    while index < 15 :
        # In表示展示的图片编号，n代表最终骰子摇到的点数，b用于判断是否摇骰子
        if In < 15 and n == 1 and b == 1:
            filename = "./images/" + str(In) + ".gif"
            img = pygame.image.load(filename)
            dispsurf.blit(img, location)
            c.tick(tick)
            In += 1
            #摇动14次之后，显示最终的点数
            if In == 14:
                filename = "./images/img1.gif"
                img = pygame.image.load(filename)
                dispsurf.blit(img, location)
                c.tick(tick)
                b = 0
                In = 1
# 显示骰子投掷的结果
def dispdiceroll(n):
    location = (710,140)
    if n == 1:
        filename = "./images/img1.gif"
        img = pygame.image.load(filename)
        dispsurf.blit(img, location)
    elif n == 2:
        filename = "./images/img2.gif"
        img = pygame.image.load(filename)
        dispsurf.blit(img, location)
    elif n == 3:
        filename = "./images/img3.gif"
        img = pygame.image.load(filename)
        dispsurf.blit(img, location)
    elif n == 4:
        filename = "./images/img4.gif"
        img = pygame.image.load(filename)
        dispsurf.blit(img, location)
    elif n == 5:
        filename = "./images/img5.gif"
        img = pygame.image.load(filename)
        dispsurf.blit(img, location)
    elif n == 6:
        filename = "./images/img6.gif"
        img = pygame.image.load(filename)
        dispsurf.blit(img, location)
```

Two functions are defined respectively to show the process and result of dice-rolling. The biggest difficulty in this part is the display of the animation, for which we have taken a number of static pictures of the throwing dice and played them in chronological order. The specific function is implemented in the diceroll function. To save space, the dice-roll function in the picture above exhibits only parts of the code.

5.4 Define events

We define three events to let the game operate as we planned. Rolling dice is used to decide how many steps the token would move. Clicking the mouse is for players to select which token to move. Finally, move button is used to judge where the token is and to accordingly control the tokens moving forward.

The most difficult function in this part is the conditional trigger after move. First, you need to judge whether the current piece reaches the end point. Secondly, you need to judge whether the piece can leave the starting point; then judge the state of the piece on the ending track; finally, judge the function of the piece after the piece has finished walking, such as placing a bomb, a star, and checking whether it touches to bombs, stars,

```
    #事件1：roll按钮被单击
    if(mouseclicked == True and mousex >= 750 and mousex <= 950 and mousey >= 500 and mousey <= 540):
        mc = 1
        n = randint(1,6)        # 获取一个0-6之间的随机数
        b=1
        mouseclicked = False    # 恢复鼠标初始的状态

        ''' 开始摇骰子'''
        diceroll(n, c, b, tick, In)
        ''' 结束摇骰子'''

        showDiceBox(n)          # 显示骰子上的数字


    #事件2：单击选择某个棋子
    elif(mouseclicked == True and mousey >= 550 and mousey <= 590):
        if(mousex >= 750 and mousex <= 790):  # 第一个棋子被选中
            movebutton = 1

        elif(mousex >= 800 and mousex <= 840): # 第二个棋子被选中
            movebutton = 2

        elif(mousex >= 850 and mousex <= 890):  # 第三个棋子被选中
            movebutton = 3

        elif(mousex >= 900 and mousex <= 940):  # 第四个棋子被选中
            movebutton = 4

        mouseclicked = False              # 恢复鼠标初始的状态
```

```
# 事件3：单击move按钮
elif(mouseclicked == True and mousex >= 750 and mousex <= 950 and mousey >= 600 and mousey <= 640):

    if(p_id == 1):      # 当前玩家1正在行动

        #如果所有的棋子都到达终点，则改变当前正在行动的玩家
        if(Player1.finalstatus[0] == 1 and Player1.finalstatus[1] == 1 and  Player1.finalstatus[2] == 1 and Player1.finalstatus[3] == 1):
            p_id = 2

        #如果有棋子没有到达终点
        else:
            if(movebutton ==1):      # 第一个棋子被选择

                #如果刚刚行动完毕，计算当前棋子的位置距离终点的距离
                if(flagIMP == 0):
                    diff = (47 - Player1.currentposition[0])
                    flagIMP = 1

                #如果当前这个棋子在待行区
                if(Player1.initialstatus[0] == 0):
                    if(n == 6 or n == 1):     # 如果骰子摇到1或者6，可以初始化棋子

                        Player1.initialstatus[0] = 1    # 记录玩家1的棋子1的初始化状态
                        initiated1[0] = position1[0]    # 将棋子放在起始的位置
                        Player1.InitiateStart(1)        # 对玩家1的棋子1进行初始化
                        mouseclicked = False            # 恢复鼠标初始的状态
                        Player1.currentposition[0] = 0  # 设定棋子1当前的位置

                    else:
                        p_id = 2                        # 如果不是1或者6，则改变当前正在行动的玩家
                        mouseclicked =False             # 恢复鼠标初始的状态
```

etc.

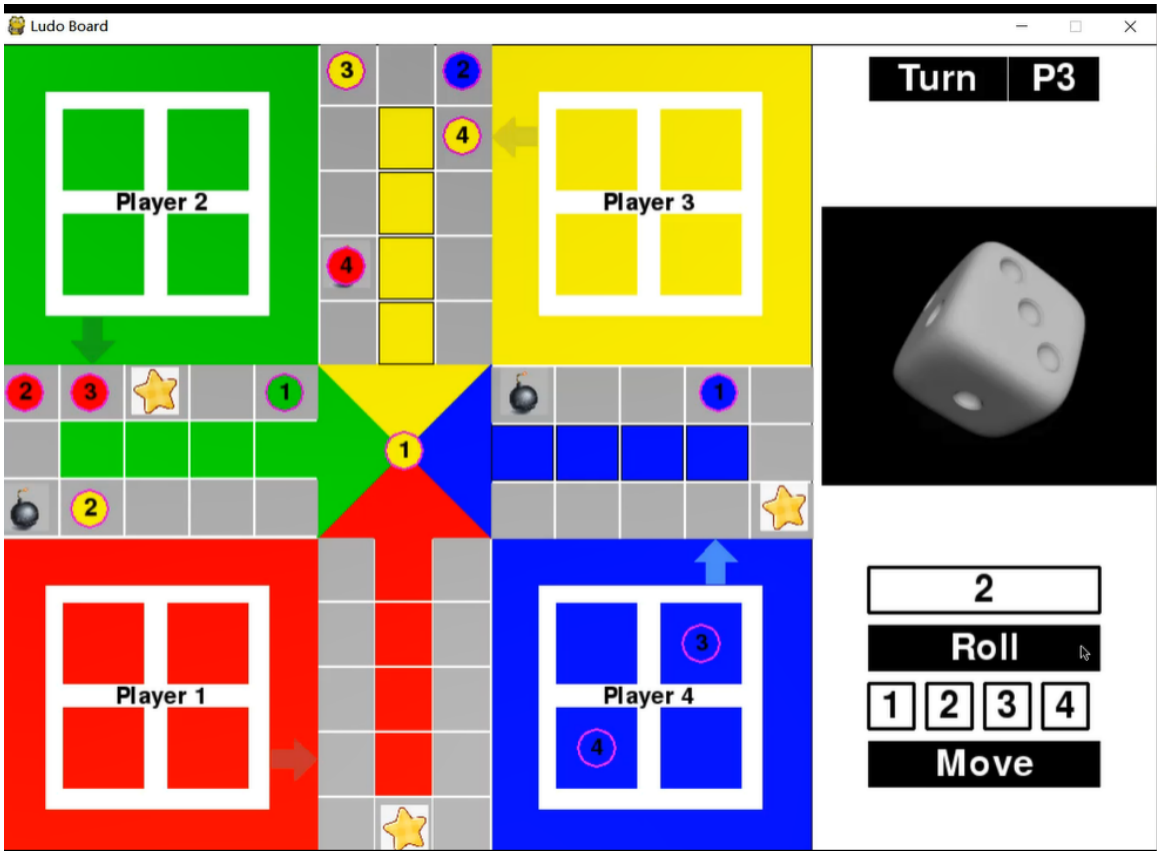
5.5 Surprises

5.5.1 Stars: skyrocketing to the end point

When players roll the dice and get a number of 1, a star is set on the chess s current position on the gameboard. The next time a token passes this star, it will be directly transferred to a place near the end point. Players encountering the star is lucky enough to predict a fast-win.

5.5.2 Bombs: return to the start point if you lose

When players roll the dice and get a number of 6, a bomb is set on the chesss current position on the gameboard. The next time a token passes this bomb, the player will has to play the **paper-scissor-rock** game with the computer. If the player loses the game, the token will be forced to fly to the start point. If the player wins, then nothing will happen and he/she will be safe.




```
# 展示炸弹位置
for Item in bomb_place:
    displaybomb(Item)

# 展示星星位置
for Item in star_place:
    displaystar(Item)
```

```
#如果前进步数已经为n步，则执行相应的触发行动
elif(i > n):
    # 如果到达的位置有炸弹，则需要玩游戏，输的时候需要将棋子退回到初始位置
    flag = False
    for bombPosition in bomb_place:
        # 当前到达的位置有炸弹
        if track1[Player1.currentposition[0]] == bombPosition:
            flag = True
            bomb_place.remove(bombPosition)  # 删除当前位置的炸弹
            game_result = Game()                    # 进行一次剪刀石头布游戏

            # 输掉游戏
            if game_result == -1:
                Player1.currentposition[0] = 0   # 棋子的位置设定为初始的位置
                Player1.backToStart()            # 将棋子放置到起点的位置

    for starPosition in star_place:
        # 当前到达的位置有星星
        if track1[Player1.currentposition[0]] == starPosition:
            flag = True
            star_place.remove(starPosition)  # 删除当前位置的星星

            Player1.currentposition[0] = 41   # 棋子的位置设定为终点轨道
            Player1.backToEnd()            # 将棋子放置到终点轨道位置

    # 如果投掷到的骰子为6，则可以安放一个炸弹
    if (n==6) and (flag == False) and (track1[Player1.currentposition[0]] != (350,350)) and (len(bomb_place)<5):
        bomb_place.append(track1[Player1.currentposition[0]])

    # 如果投掷到的骰子为1，则可以安放一个星星
    if (n==1) and (flag == False) and (track1[Player1.currentposition[0]] != (350,350)) and (len(star_place)<5):
        star_place.append(track1[Player1.currentposition[0]])

    p_id = 2 #重置当前行动的棋子
    i=1 #重置已经向前的步数
    mouseclicked = False # 恢复鼠标初始的状态
```

6 Problems and optimization

6.1 Lack of enough accuracy

In the part of rock-paper-scissors game, we used openCv for gesture recognition. But in many cases, the success rate of recognition is related to the quality of the image and the cleanliness of the background. Recognition is only successful when the background is white. Sometimes it takes a refresh or trial and error to succeed.

6.2 Problems with game props

Props in the game may appear in the same grid in many times, which may cause the game to crash. The overlap of the pieces in the same lattice will not be shown on the board. It's not very gamer friendly.

6.3 Playability of the game

The game is highly repetitive in its steps, which can be annoying to the player. The ultimate success of the game also depends on the luck of the players. Just two items may not please the player.

7 Reference

During the project completion process, we looked at GitHub's open source code for image identification and modified it to be added to the Ludo game.

Address:
https://github.com/lzane/Fingers-Detection-using-OpenCV-and-Python/blob/master/new.py

8 Our Github Website

Address:
https://github.com/chengboyuan/Human-computer-Interaction-Ludo-Game-Based-on-Image-Recognition