جامعة دمشق
كلية الهندسة المعلوماتية
قسم الذكاء الصنعي
نادي الروبوتية

IQ_Block_Puzzle

مشروع للمشاركة في مسابقة الأردوينو لطلاب النادي عام 2024

الطلاب :

طارق الصباغ التاجي ماهر سليمان

المدرب:

خلدون عطايا

2024–2025
1446–1447

الملخص:

يهدف هذا المشروع لحل ال IQ_Puzzle من فئة (8*8) باستخدام بعض خوارزميات البحث الذكية حيث تم دمج بعض التقنيات الحديثة في المشروع من خلال التقاط صورة للرقعة ويتم معالجتها باستخدام مكتبة opencv-python

يبدأ المشروع بالتقاط صورة الرقعة ويقوم بمعالجتها وفصل الألوان والقطع لتحديد شكل الرقعة الاولي ومحاولة إيجاد الحل باستخدام بعض الخوارزميات
فتكون النتائج النهائية لهذا المشروع الحصول على الرقعة مكتملة الحل بأحد اشكال حلولها الكبيرة بالإضافة الى انه من الممكن تعديل على هذه الفكرة وتطويرها بشكل اكبر...

جدول المحتويات:

الفصل الأول :

توصيف المشروع :

يقوم المشروع بحل رقعة ال IQ_Puzzle بوضع المستخدم هذه الرقعة باي شكل من الاشكال ثم يتم إعطاءه احد حلولها في البداية قمنا بعملية بحث عن مشروع مشابه في الانترنت فلم نجد لذا بدأنا بتعلم بعض خوارزميات البحث مثل (BFS,DFS,A*) ولكن وجدنا انه من الصعب تطبيق الخوارزميات المذكورة في إيجاد الحل .

الطريقة المستخدمة :

قمنا باستخدام خوارزمية brute force and backtracking أي تجربة جميع الحالات التي يمكن ان توضع القطعة ضمن الرقعة ولكن واجهتنا مشكلة في التعقيد الزمني للخوارزمية حيث انها استغرقت حوالي 3 ساعات بحثا عن الحل ولم تصل الى حل ف قررنا استخدام بعض الخوارزميات لتحسين التعقيد وقمنا باستخدام خوارزمية ال DFS لرفض بعض الحالات الغير ممكنة لخفض التعقيد الزمني بشكل اكبر

Brute Force

DFS

تم استخدام أيضا مكتبة ال opencv لمعالجة صورة الرقعة وتحويلها الى مصفوفة لتقوم الخوارزمية بإيجاد الحل :

```
pip install  opencv-python
```

تم استخدام مكتبة ال pygame تعرض واجهة تنفيذ الكود خطوة خطوة

```
pip install pygame
```

الفصل الثاني :

مراحل تطوير و تطبيق المشروع:

1.التحليل والتخطيط :

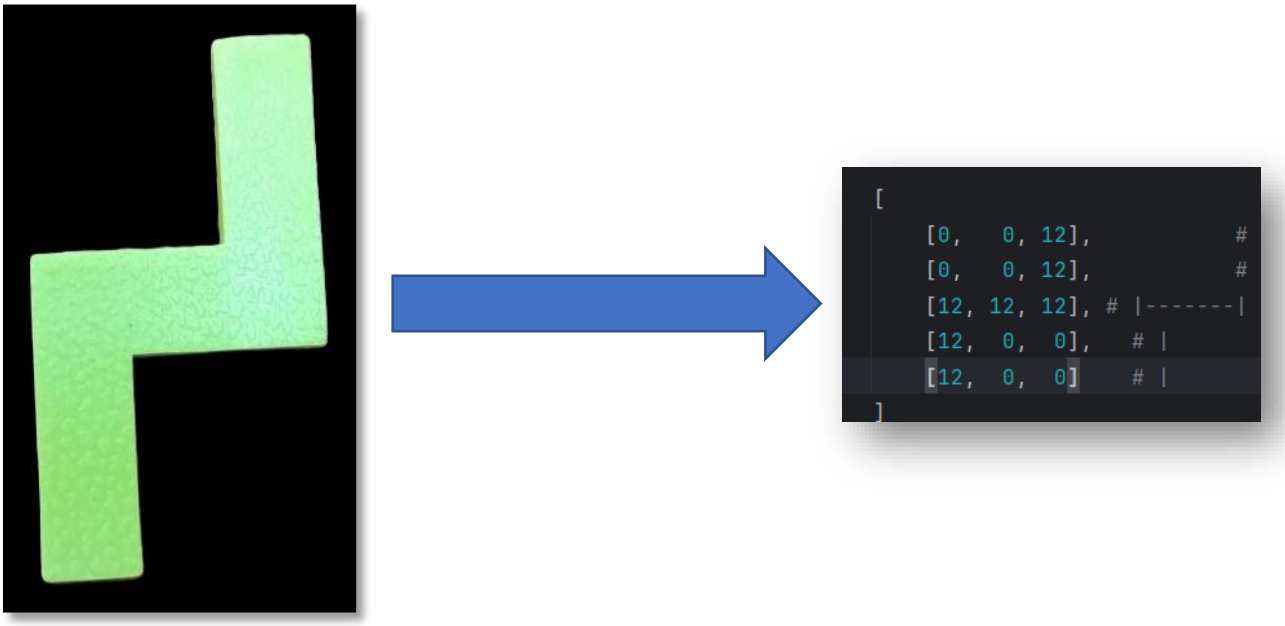
في هذه المرحلة يتم عمل دراسة لجدوى المشروع لفهمه والتحقق من الامكانيات لتنفيذه خلال الفترة الزمنية المتاحة وتحديد متطلبات المشروع(Requirements) (موارد بشرية, ميزانية, أدوات برمجية)
يتضمن التقرير الناتج عن مرحلة التحليل ما يلي:

- أدوات برمجية ومكاتب لتطبيق المشروع
- كاميرا وأدوات ميكانيكية

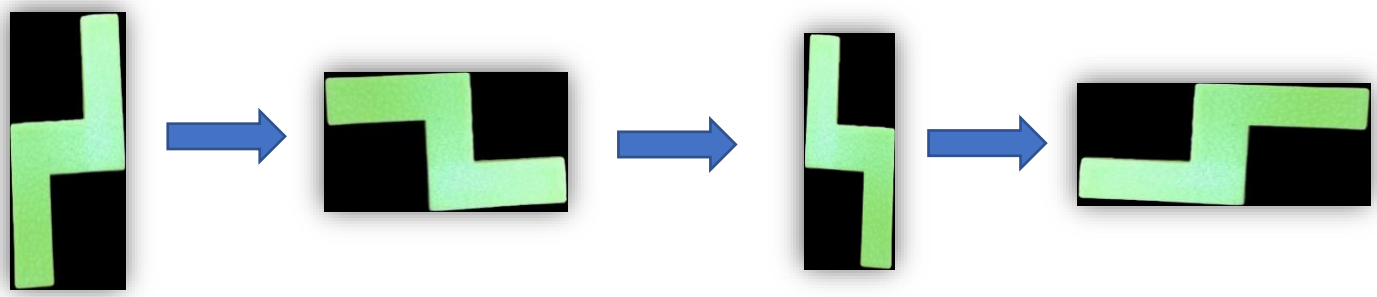
2. البرمجة :

1) الخوارزمية:

- قمنا بتحويل اشكال القطع الى مصفوفات وكل قطعة لها رقم خاص بها :



- قمنا بإيجاد جميع الوضعيات الممكنة لوضع القطعة على الرقعة:



❖ وهنا بدأنا بكتابة الخوارزمية التي تعتمد على خوارزميتي(brute force and backtracking algorithm):
آلية عمل الخوارزمية: نأخذ اول قطعة مدخلة بأول وضعية ثم تتحقق من إمكانية وضعها بحيث لا نضعها فوق قطعة أخرى او خارج الرقعة ا اذا كان من الممكن وضعها نضيفها الى الرقعة والا نجربها بوضعيات أخرى وهكذا نعيد العملية من اجل القطعة 2 و 3 و 4 إلخ..... الى ان نصل الحل:

```python
def runner(allPieces, board):  4 usages  new *
    global stop
    if stop:
        # test_GUI.draw(board)
        return

    if all([all(row) for row in board]):
        print("Puzzle solved!")
        print_matrix(board)
        stop = True
        return

    if not allPieces:
        print("No more pieces left to place.")
        return

    pieces = allPieces[0]

    for i, piece in enumerate(pieces):
        legal_moves = get_legal_squares(board, piece)

        for x, y in legal_moves:
            newBoard, valid = add_piece(board, piece, x, y)
            if (optemization(newBoard)):
                if valid:
                    if not stop:
                        test_GUI.draw_with_fade(newBoard, piece, x, y)
                        # pygame.time.delay(test_GUI.delay_duration)
                        runner(allPieces[1:], newBoard)
```

❖ وأخيرا قمنا بإضافة بعض االتحسينات من خلال استخدام خوارزمية ال DFS التي تقوم بحذف بعض الحالات التي لا توصل الى حل :

```python
def DFS(board,visited,r,c):  5 usages  new *
    if r < 0 or c < 0 or r >= 8 or c >= 8 or visited[r][c] or board[r][c] != 0:
        # print(count)
        return 0
    visited[r][c] = True
    count=1
    count+=DFS(board,visited,r + 1, c)
    count+=DFS(board,visited,r - 1, c)
    count += DFS(board,visited,r, c + 1)
    count+=DFS(board,visited,r, c - 1)
    return count
```

2) الابصار:

باستخدام مكتبة ال opencv-python:

- قمنا بالتقاط صورة الرقعة ثم قمنا بإيجاد الكونتور الأكبر في الصورة (لايجاد موضع الرقعة بالضبط):

```
def takePhoto():  new *
    cap=cv2.VideoCapture(2)
    last_time=time.time()
    stop=2
    path="board.jpg"
    while True:
        ret,frame=cap.read()
        # cv2.imshow("frame",frame)
        # if cv2.waitKey(1) & 0xFF == d
        #     break
        if time.time()-last_time>=stop:
            cv2.imwrite(path,frame)
            return frame
```

- وبعدها تم تعديل سطوع الصورة لتوضيح الألوان وتجنب مشاكل الإضاءة ثم تم تقسيم الصورة الى 64 صورة (جعلها ك مصفوفة 8*8):

```
img_c = imgWarpColored.copy()
img_c = img_c[25:425, 25:425]
brightness =3
# Adjusts the contrast by scaling the pixel values by 2.3
contrast = 1.3
img_c = cv2.addWeighted(img_c, contrast, np.zeros(img_c.shape, img_c.dtype), beta: 0, brightness)
parts=[]
vertical_splits = np.vsplit(img_c, indices_or_sections: 8)
for v_split in vertical_splits:
    horizontal_splits = np.hsplit(v_split, indices_or_sections: 8)
```

- معالجة الصورة لاستخراج الرقعة منها :

```
def getBoard(board,parts):  1 usage  new *
    j=0
    s=0
    valid=False
    global indexPices
    for index, part in enumerate(parts):
        for i, (lower, upper) in enumerate(color_ranges):
            hsv_image = cv2.cvtColor(part, cv2.COLOR_BGR2HSV)
            mask = cv2.inRange(hsv_image, lower, upper)
            contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            if contours:
                for contour in contours:
                    area=cv2.contourArea(contour)
                    if (area > 1000):
                        if i+1 not in indexPices:
                            indexPices.append(i+1)
                        board[s][j]=i+1
                        # print(area)
                        # color_num.append(i + 1)
                        # cnt=cnt+1
                        approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, closed: True), closed: True)
                        cv2.drawContours(part, contours: [approx], -1, color: (0, 255, 0), thickness: 2)
```

❖ حذف القطع المستخدمة في البورد:

```python
def get_piece(indexPices):  1 usage  new *
    global new_pices
    # print(indexPices)
    for index in indexPices:
        new_list.append(pices[index-1])
    for pice in pices:
        if pice not in new_list:
            new_pieces.append(pice)

    return new_pieces
```

3) الواجهات:

باستخدام مكتبة pygame :

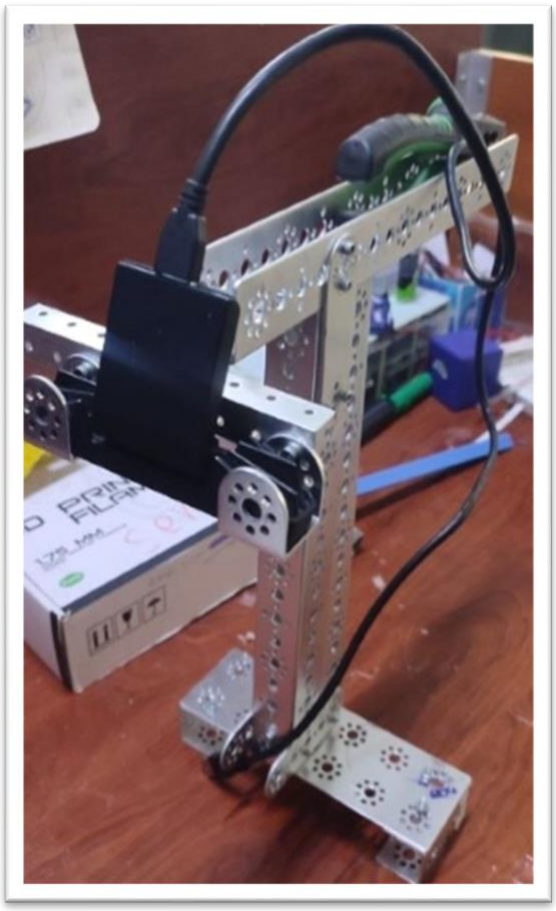
❖ عرض الرقعة وعرض تحرك كل قطعة في الرقعة :

```python
def draw(board):  1 usage  new *
    screen.fill((255, 255, 255))

    # رسم الشبكة
    for row in range(8):
        for col in range(8):
            color = colors[board[row][col]]
            pygame.draw.rect(screen, color,  rect: (col * cell_size, row * cell_size, cell_size, cell_size))

            # خطوط الشبكة
            pygame.draw.rect(screen,  color: (0, 0, 0),  rect: (col * cell_size, row * cell_size, cell_size, cell_size),  width: 1

    pygame.display.flip()
```

```python
def draw_with_fade(board, piece, x, y):  1 usage  new *
    """ إضافة تأثير الاختفاء عند رسم القطعة """
    fade_steps = 10  # عدد مراحل التلاشي
    for step in range(1, fade_steps + 1):
        screen.fill((255, 255, 255))

        for row in range(8):
            for col in range(8):
                if x <= row < x + len(piece) and y <= col < y + len(piece[0]) and piece[row - x][col - y] != 0:
                    # تعديل لون القطعة أثناء التلاشي
                    color = colors[piece[row - x][col - y]]
                    faded_color = (color[0], color[1], color[2], int(255 * (step / fade_steps)))
                    pygame.draw.rect(screen, faded_color,  rect: (col * cell_size, row * cell_size, cell_size, cell_size))
                else:
                    color = colors[board[row][col]]
                    pygame.draw.rect(screen, color,  rect: (col * cell_size, row * cell_size, cell_size, cell_size))

                pygame.draw.rect(screen,  color: (0, 0, 0),  rect: (col * cell_size, row * cell_size, cell_size, cell_size),  width: 1

        pygame.display.flip()
        pygame.time.delay(fade_duration // fade_steps)
```

مرحلة الميكانيك :

ميكانيك بسيط جدا يعتمد على كيتات التيتركس لانشاء ستاند لوضع الكاميرا عليه نوع الكاميرا **Creative**:



الفصل الثالث :

خطوات تطبيق المشروع:

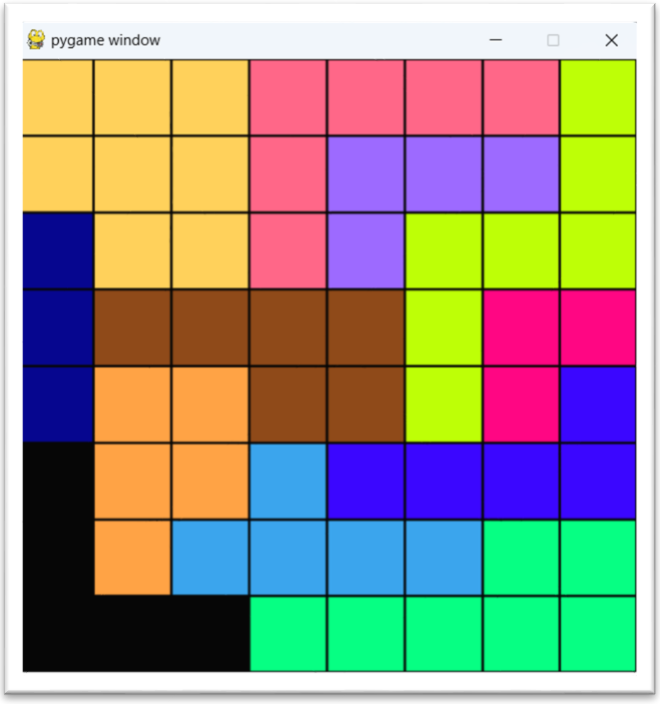
مثال عن تطبيق المشروع:

1. التقاط الصورة من الكاميرا :

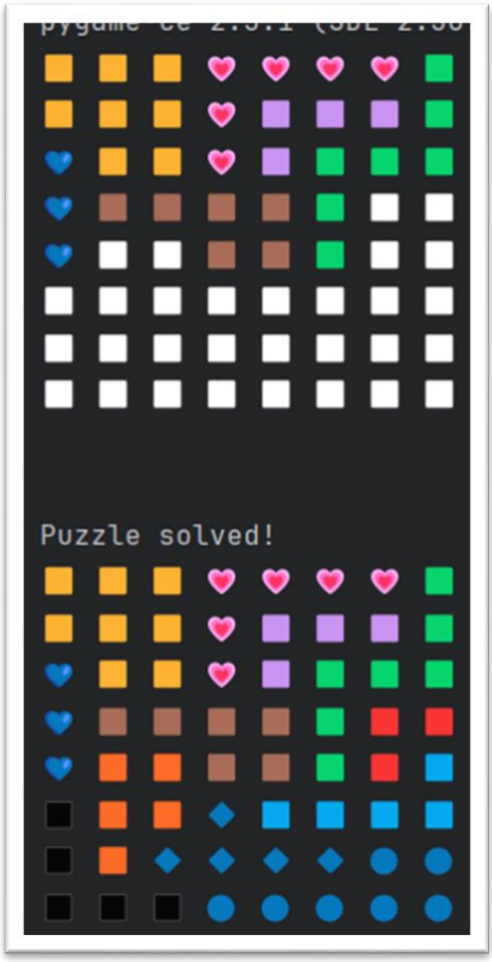
2. إيجاد الرقعة بعد تحسين الألوان :

3. تقسيم الصورة لإيجاد الرقعة من خلال ألوان القطع:

4. الواجهة التي تقوم بإظهار الحل:



5. طباعة الرقعة الأولية والنهائية على ال console :



6. رابط الكود:

CODE

الفصل الرابع : الخاتمة

الآفاق المستقبلية :

لا يزال المشروع في بدايته وهو بحاجة الى إضافات تعديلات وتحسينات كثيرة عليه برمجيا وعتاديا من خلال إيجاد خوارزميات افضل للوصول الى الحل او تطوير الفكرة

فكما تمّ التوضيح، يملك هذا المشروع الكثير من الآفاق المستقبلية التي بدأ البحث والعمل على تنفيذ بعضها، وبعض الآفاق الاخرى التي تحتاج تحديثا استخدام عتاديات جديدة واستبدال .

ويُلاحظ من خلال الاجراء العملي للمشروع ان الخوارزمية حتى الان قادرة على إيجاد حل لبعض الرقع التي تحوي على 4 قطع على الأقل الا ان الخوارزمية ليست فعالة بالمجمل

وكان من المخطط في بداية المشروع استخدام بعض الخوارزميات السريعة كما ذكرنا في المقدمة وذلك لجعل الكود اكثر فعالية واكثر سرعة ولكن تم التخلي عن الفكرة نتيجة ضعف الخبرة وقلة الوقت المتاح في تقديم هذه المشروع

الخطوة التالية :

إيجاد حل روبوتي (ميكانيكي) يوضح حل هذا اللعبة ولكن بشكل ملموس و أقرب إلى الواقع

الحلول الروبوتية الممكنة :

1.محور X,Y,Z مزود ب (reversed air flow nozzle) :

محور مثبت فوق اللعبة, حدوده هي حدود اللعبة له رأس يسحب الهواء كي يتمكن من إمساك القطع الصغيرة ويضعها في مكانها الصحيح

في الصورة تقنية مشابهة جدا للتقنية التي فكرنا فيها
Mark Rober's robotic arm solution for puzzle

2.ذراع روبوتية:

ذراع روبوتية تقوم بالتقاط القطع وتدويرها أو قلبها لوضعها في المكان المناسب

3. RGB LED Matrix:

أما بالنسبة للحل الذي اخترناه في النهاية فهو ربط مشروعنا والخوارزمية ب (RGB LED Matrix 8*8)

ولكن واجهتنا بعض المشاكل مثل السعر المبالغ فيه لهذه القطعة الإلكترونية و عدم توافرها بالسوق المحلية على الرغم من ذلك لجأنا إلى حل أخير بمتناول الأيدي وهو محاكاة هذه الطريقة على برنامج محاكاة متقدم (Proteus), وهذا الحل الذي يتم العمل عليه حاليا