人工智慧
x
Puzzle Game

106590016 游明憲	106590023 徐紹崴	106590024 龍育達
106590029 劉聿彤	106590034 吳陽生	106590035 郭宗育
106590037 劉聰池	106590040 温致綱	指導老師 郭忠義

01 Introduction
遊戲簡介

02 Graphical User Interface
圖形化介面

03 Algorithm
演算法

04 Demo

01

遊戲簡介

簡介

主頁面

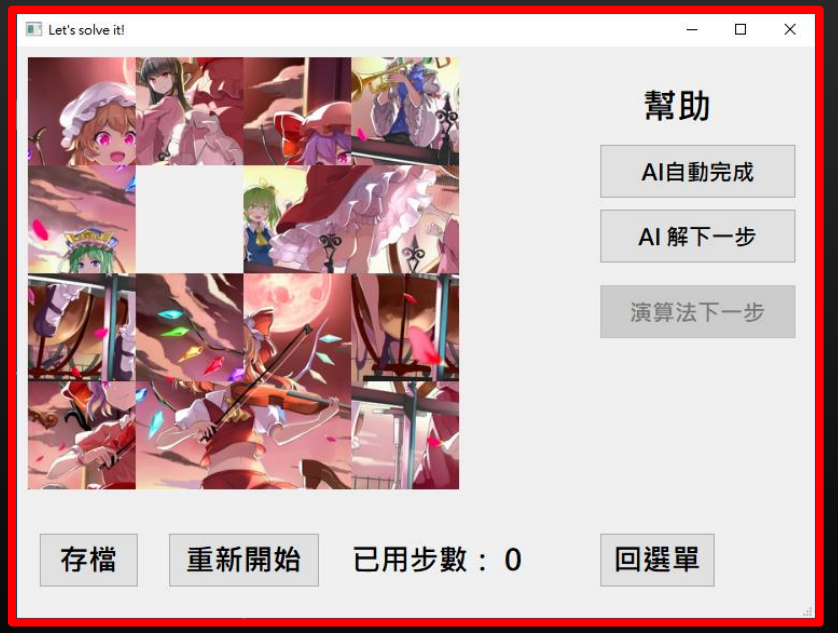
- 自由選擇圖片
- 自由調整行/列數
- 讀取存檔

準備頁

- 自由選擇空格開始位置

遊戲頁

- AI、演算法幫助
- 手動遊玩
- 存檔



02

圖形化介面

圖片分割

```python
def cropImage(self, image, cropNum):
    width, height = image.size
    item_width = int(width / cropNum)
    box_list = []
    for i in range(0, cropNum):
        for j in range(0, cropNum):
            box = (j * item_width, i * item_width, (j + 1) * item_width, (i + 1) * item_width)
            box_list.append(box)
    image_list = [image.crop(box) for box in box_list]
    self.save_img(image_list)
    pixmapList = [self.ConvertPILtoPixmap(img) for img in image_list]
    return pixmapList
```

借助python的pillow的函式庫，
得以輕鬆的以crop，搭配box獲取指定區域的圖片

檔案讀寫

```python
def writeJson(savePath, obj):
    print(json.dump(obj, open(savePath, "w"), cls=UserJSONEncoder))
    print("write json")


def readJson(loadPath):
    jsonData = json.load(open(loadPath))
    print("load json")
    return jsonData
```

由於 python的字典，與 json格式的轉換容易，
我們能精簡的用dump 將字典轉換成 json格式；load 則反之，
引此我們選用json來做紀錄檔的存取之媒介。

信號

```
class Signal(QtCore.QObject):
    signal = QtCore.pyqtSignal(str)

    def __init__(self):
        super().__init__()

    def Shoot(self, message):
        self.signal.emit(message)
```

△定義信號

```
self.data.dataSignal.Shoot("Goto3")
```

△發射信號

```
def ReceiveMessage(self, message):
    if message == "Goto3":
        print("Receive! " + message)
        self.ClearWindowsData()
        self.CreateRandomPuzzle()
```

△接收信號

```
self.data.dataSignal.signal.connect(self.ReceiveMessage)
```

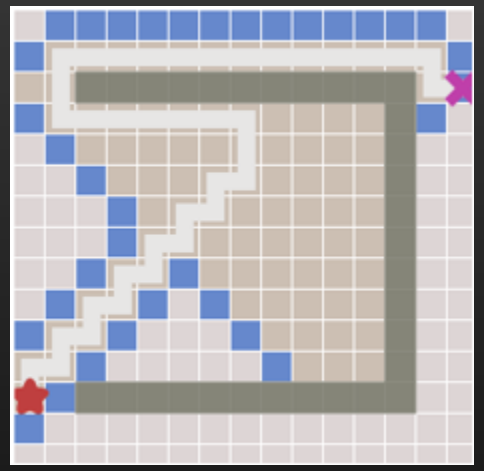
△綁定信號

PyQt有一種獨特的方式，讓不同的視窗間可以溝通，那就是signal。

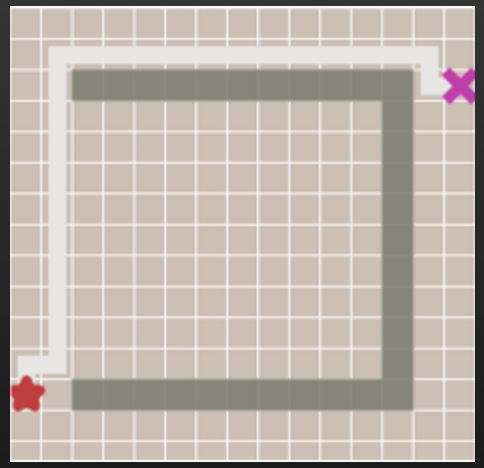
03

演算法

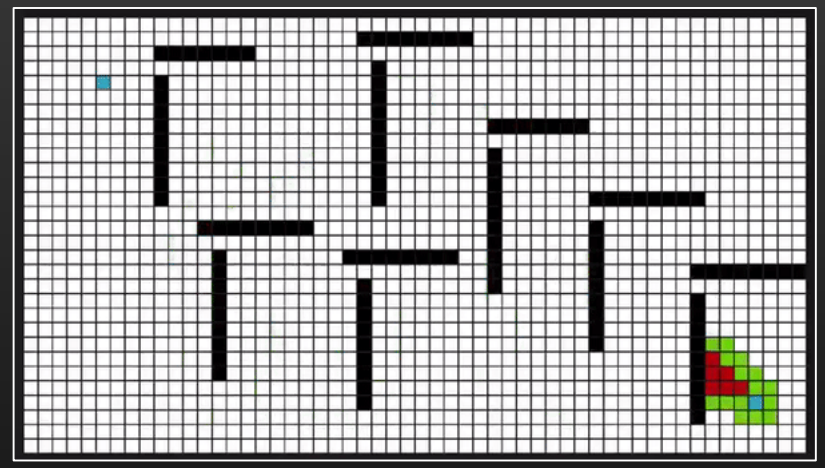
演算法 - 蓋覽



- 最佳優先搜尋



- 廣度優先搜尋



- A*

演算法 - A*

```
cur = self.open[0]
while(len(self.open) > 0):
    cur = self.open[0]
    if(self.h(cur.data,goal) == 0):
        break
    for i in cur.generate_child():
        i.parent = cur
        i.fval = self.f(i,goal)
        if self.stateInStates(i,self.closed):
            continue
        if not self.stateInStates(i,self.open):
            self.open.append(i)
        if self.stateInStates(i,self.open) and  i.fval < self.GetStateFromStates(i,self.open).fval:
            self.open.remove(self.GetStateFromStates(i,self.open))
            self.open.append(i)
    self.closed.append(cur)
    del self.open[0]
    """ sort the opne list based on f value """
    self.open.sort(key = lambda x:x.fval,reverse=False)
```

g = 起始狀態到目前狀態走了幾步(啟發式函數)

h = 目前狀態與最終狀態的曼哈頓距離總和

cost = g + h(對應程式碼內的 astar函數)

1. 建立一個open list(存放從close list拓展出來的狀態)和close list(存放走過的狀態)
2. 將初始狀態加入open list中
3. 若是open list的數量 > 0，則代表已經找到結果
4. 將open list中cost最小的狀態設為current state，並將此狀態從open list中移除並加入close list中
5. 尋找可以從current state拓展出去的狀態並加入neighbor list(每次會重置)中
6. 尋訪每個neighbor list中的狀態，若是此狀態不在open list中，或是在open list中，但由這個狀態拓展出去的cost比原先在open list中的cost小的話，則將此狀態新增(覆蓋)到open list中
7. 回到3

演算法 - BFS

```
#判斷可否向上
def CanUp(state, spaceOrder):
    return (spaceOrder - gameTypeNum) >= 0
```

```
#BFS
currentNode = Node(None, srcOrder)
count = 0
while currentNode.state != destOrder:
    for neighbor in FindNeighbor(currentNode):
        nodes.put(neighbor)
    currentNode = nodes.get()
    count = count + 1
```

演算法 - DFS

```
#DFS
currentNode = Node(None, srcOrder)
count = 0
stack = []
stack.append(currentNode)
isInStack = False
while currentNode.state != destOrder:
    if(len(currentNode.child) == 0 and not currentNode.searched):
        currentNode.child = FindNeighbor(currentNode)
    if(len(currentNode.child) != 0):
        if(len(currentNode.child) == 1):
            currentNode.searched = True
        currentNode = currentNode.child.pop()
    for i in stack:
        if (i.state == currentNode.state):
            isInStack = True

    if (isInStack):
        currentNode = currentNode.parent
    else:
        stack.append(currentNode)
    if currentNode.searched:
        currentNode = currentNode.parent
        stack.pop()
    print(count)
    count = count + 1
    isInStack = False
```

04

Demo

參考資料

- **A* Pathfinding (E01: algorithm explanation)**
  https://www.youtube.com/watch?v=-L-WgKMFuhE
- **樹搜尋、回溯與分支定界演算法**
  http://slidegur.com/doc/150452/alg4-1203-nomark-
- **Pillow - Image Module**
  https://pillow.readthedocs.io/en/stable/reference/Image.html
- **PyQt5 - Reference Guide**
  https://www.riverbankcomputing.com/static/Docs/PyQt5/

Thank You