

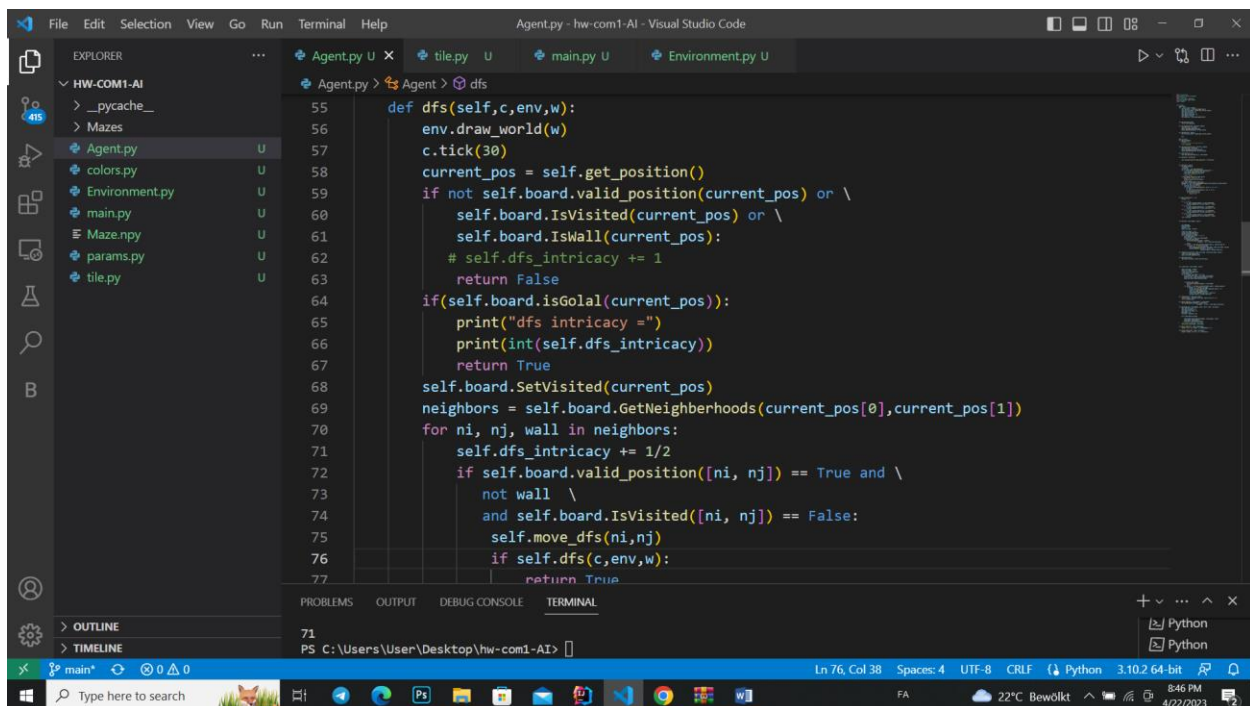
به نام خدا

تمرین اول کامپیوتری هوش مصنوعی

پدرام رمضان زاده

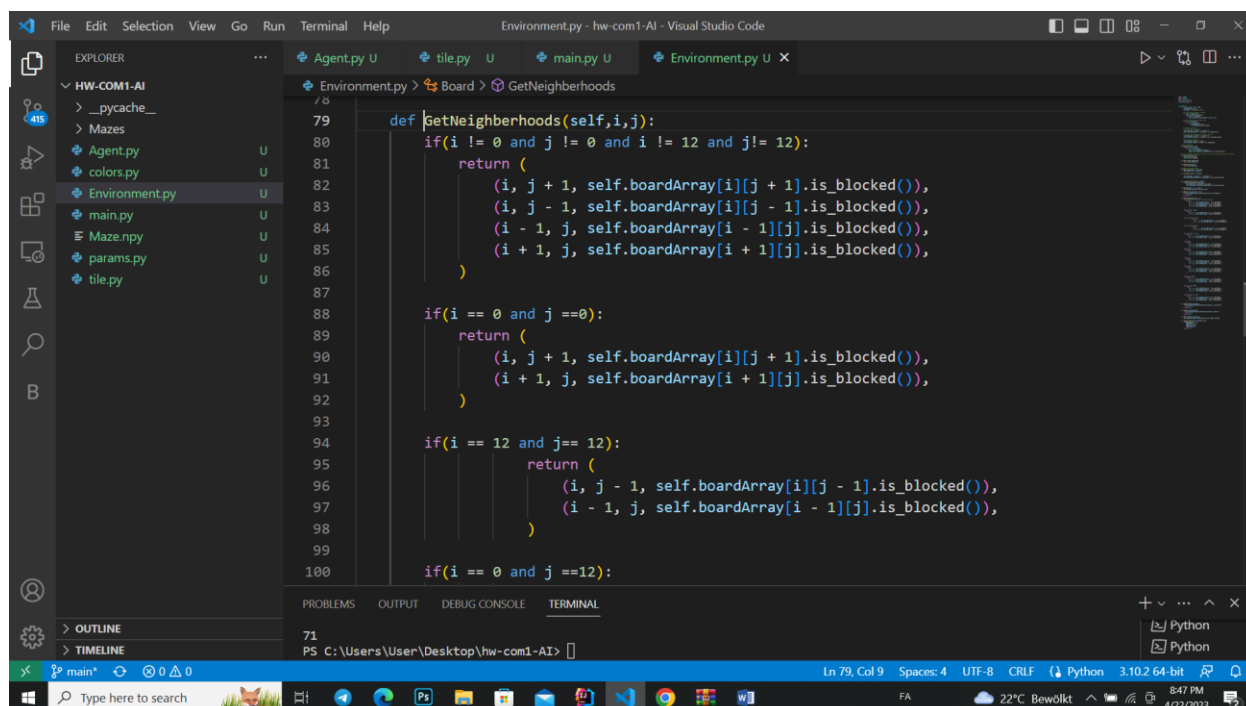
شماره دانشجویی: ۹۹۲۴۳۰۸۵

برای مسیریابی با استفاده از الگوریتم DFS تابع dfs را به صورت بازگشتی پیاده سازی شده است. به این صورت بررسی میکند که اگر در مسیرش دیوار نبود و مسیر معتبر بود به سراغ تابع بعدی که GetNeighborhoods است میرود و در آن همسایه ها را برای انتخاب مسیر چک میکند.



```
55 def dfs(self, c, env, w):
56     env.draw_world(w)
57     c.tick(30)
58     current_pos = self.get_position()
59     if not self.board.valid_position(current_pos) or \
60         self.board.IsVisited(current_pos) or \
61         self.board.IsWall(current_pos):
62         # self.dfs_intracacy += 1
63         return False
64     if(self.board.isGoalal(current_pos)):
65         print("dfs intricacy =")
66         print(int(self.dfs_intracacy))
67         return True
68     self.board.SetVisited(current_pos)
69     neighbors = self.board.GetNeighborhoods(current_pos[0],current_pos[1])
70     for ni, nj, wall in neighbors:
71         self.dfs_intracacy += 1/2
72         if self.board.valid_position([ni, nj]) == True and \
73             not wall \
74             and self.board.IsVisited([ni, nj]) == False:
75             self.move_dfs(ni,nj)
76             if self.dfs(c,env,w):
77                 return True
```

در تابع GetNeighborhoods تمامی حالات ممکن برای حرکت چک میشود.



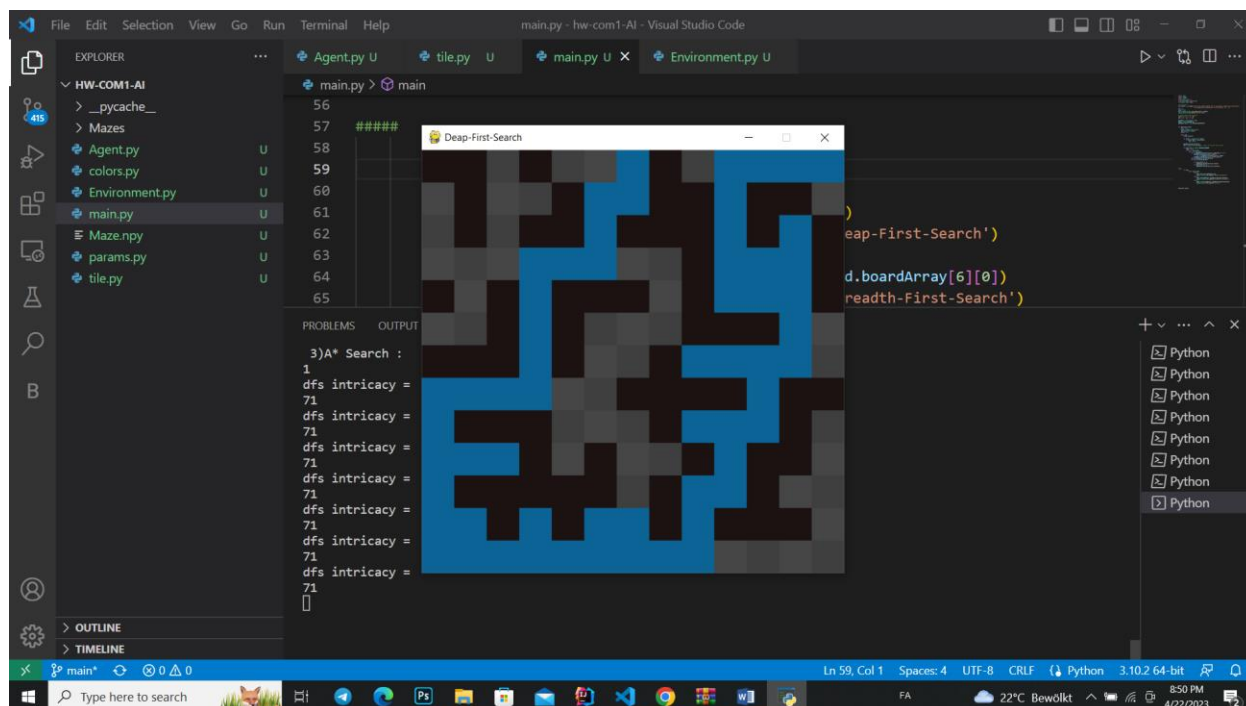
```
def GetNeighborhoods(self,i,j):
    if(i != 0 and j != 0 and i != 12 and j!= 12):
        return (
            (i, j + 1, self.boardArray[i][j + 1].is_blocked()),
            (i, j - 1, self.boardArray[i][j - 1].is_blocked()),
            (i - 1, j, self.boardArray[i - 1][j].is_blocked()),
            (i + 1, j, self.boardArray[i + 1][j].is_blocked()),
        )

    if(i == 0 and j ==0):
        return (
            (i, j + 1, self.boardArray[i][j + 1].is_blocked()),
            (i + 1, j, self.boardArray[i + 1][j].is_blocked()),
        )

    if(i == 12 and j== 12):
        return (
            (i, j - 1, self.boardArray[i][j - 1].is_blocked()),
            (i - 1, j, self.boardArray[i - 1][j].is_blocked()),
        )

    if(i == 0 and j ==12):
```

در این نوع جستجو نیازی به دیدن تمامی خانه ها نیست. بنابراین پیچیدگی این الگوریتم کم تر است اما مسیر بهینه را به ما نمیدهد.



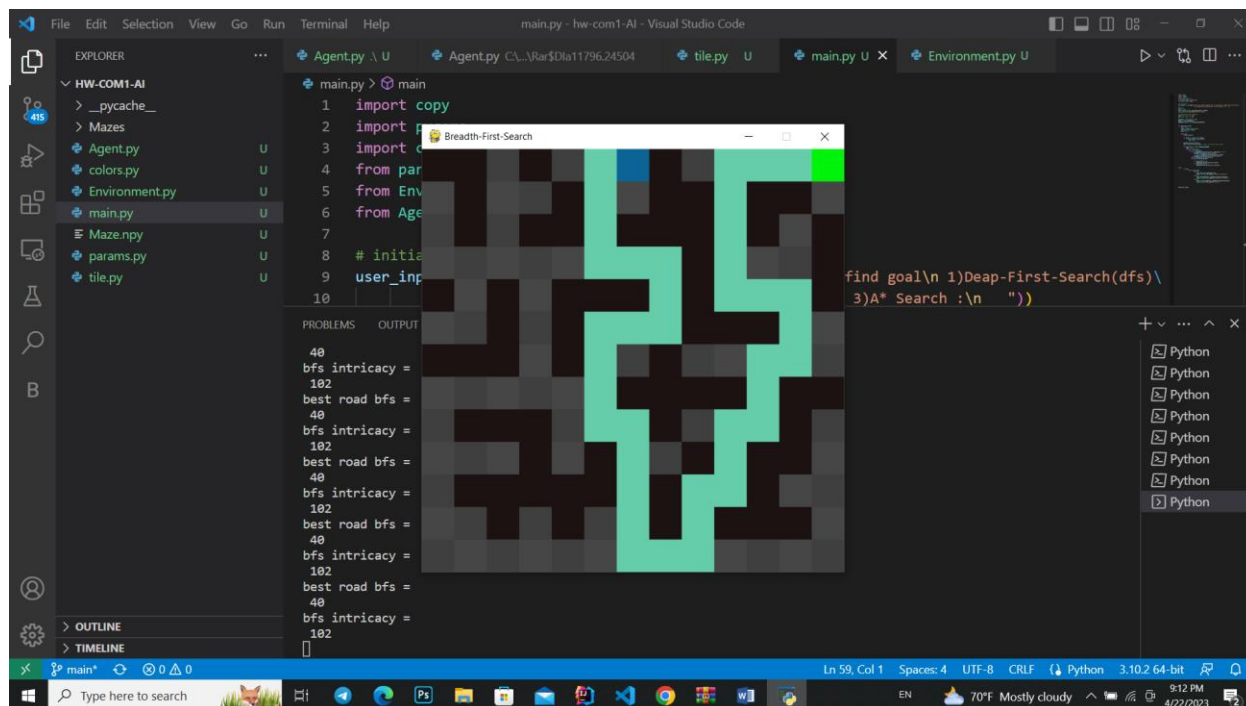
```
main
56
57 #####
58
59
60
61
62
63
64
65
```

PROBLEMS OUTPUT

```
3)A* Search :
1
dfs intricacy =
71
dfs intricacy =
71
dfs intricacy =
71
dfs intricacy =
71
dfs intricacy =
71
dfs intricacy =
71
dfs intricacy =
71
```

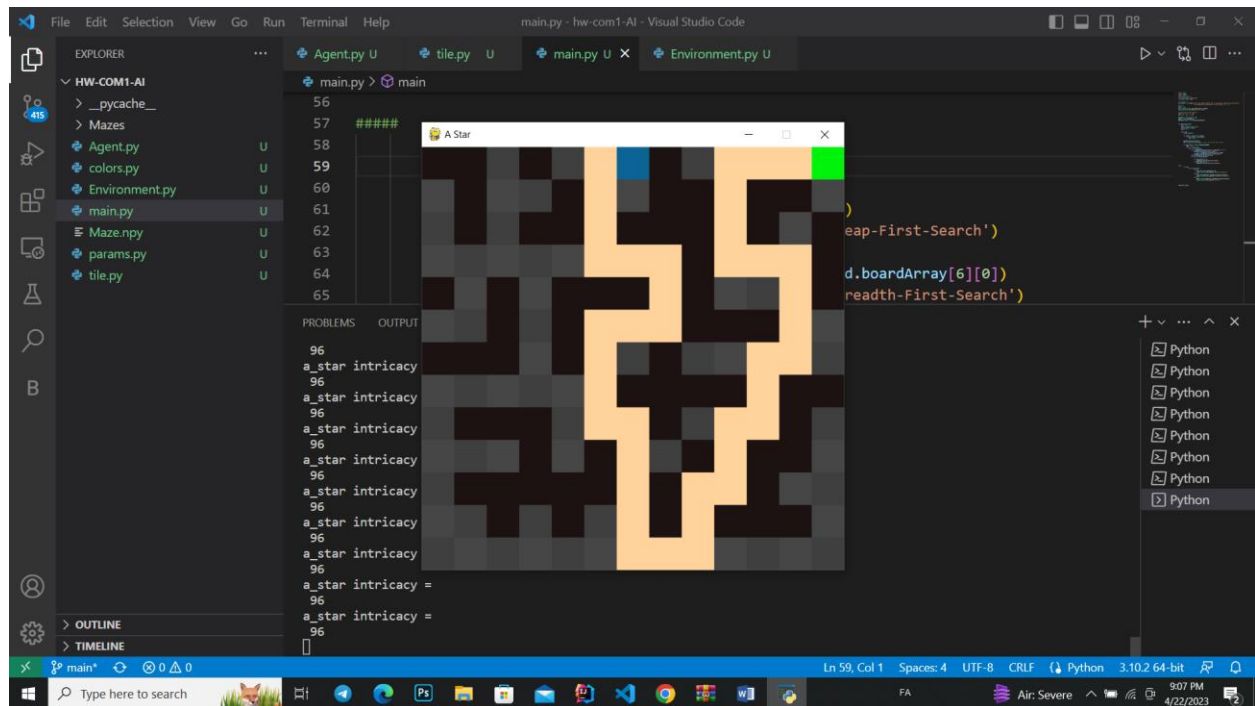
همان طور که در شکل بالا مشاهده میشود با دیدن ۷۱ خانه مسیر را پیدا کرد که بهترین مسیر نیست.

برای bfs به خاطر اینکه باید خانه های بیشتری رو ببینیم از دو تا لیست استفاده کردم که توی یکی نودهای دیده شده رو میزاریم و توی یکی دیگه نود های دیده نشده. حالا به حلقه میزنیم و این قدر ادامه میدیم تا لیست نود های دیده نشده تموم بشه.



پیچیدگی زمانی این الگوریتم ۱۰۲ میشود یعنی از dfs بیشتر شده اما مسیر کوتاه تری رو داره نشون میده.

الگوریتم a star هم مثل bfs پیاده سازی کردم با این تفاوت که به تابع h اضافه شده که فاصله نقطه شروع تا هدف رو در نظر میگیره .



مشاهده میشود که در الگوریتم bfs و a star به یک مسیر میرسیم با این تفاوت که در a star با دیدن خانه های کم تری به نتیجه دلخواه میرسیم.

پس پیچدگی زمانی به صورت زیر شد:

$BFS > A^* > DFS$

و طول مسیر به صورت زیر شد:

$A^* = BFS < DFS$