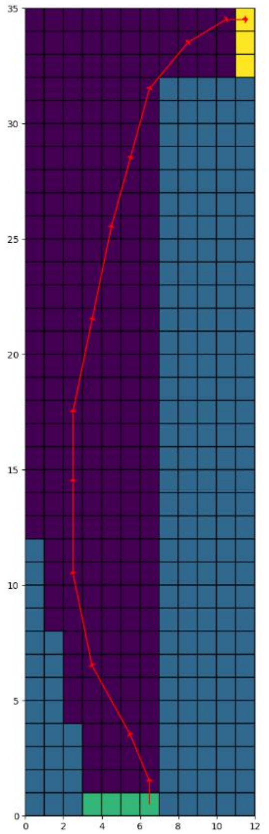
深蓝学院
shenlanxueyuan.com

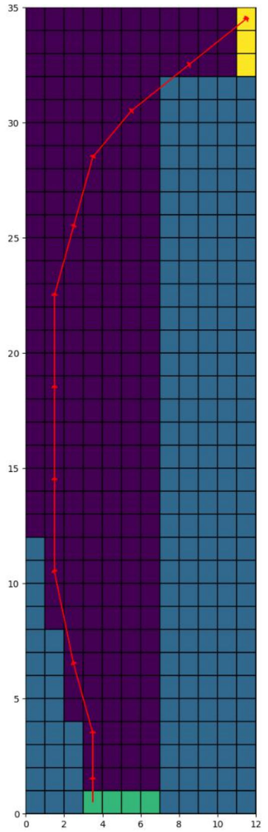
# 第7章作业分享

主讲人 ******

# 问题定义

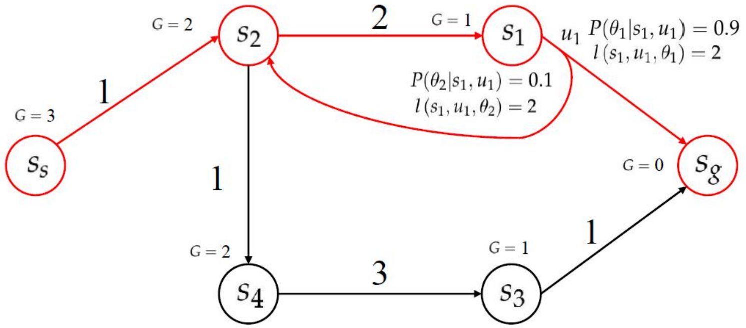
- 这里我们需要规划从START_LINE的随机一点开始到FINISH_LINE任意一点结束的路径
- 我实现的时候对问题进行了一下拓展，要求可以在FINISH_LINE任意一点时可以停下来，即速度为 0
- 左图的结果可以在FINISH_LINE停下来
- 右图的结果只保证能够到FINISH_LINE，但下一秒可能冲出去



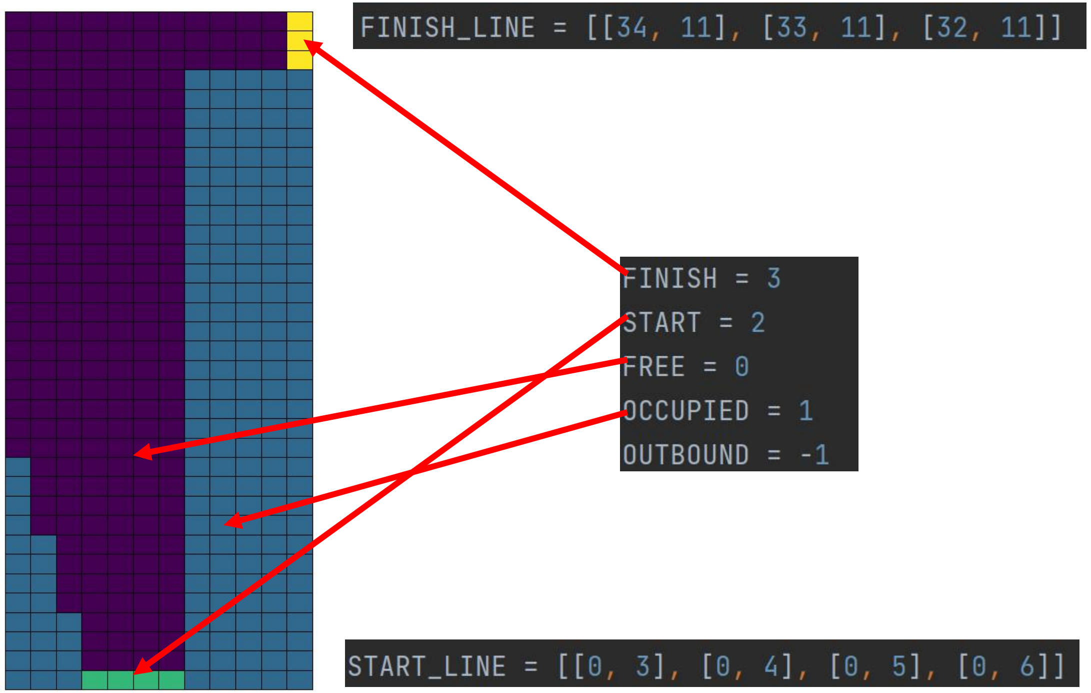


# RTDP 算法回顾

- RTDP algorithm:
  1. Initialize $G$ values of all states to admissible values;
  2. Follow greedy policy picking outcomes at random until goal is reached;
  3. Backup all states visited on the way;
  4. Reset to $x_s$ and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;



## Track格子的类型

- 

```
FINISH_LINE = [[34, 11], [33, 11], [32, 11]]
```

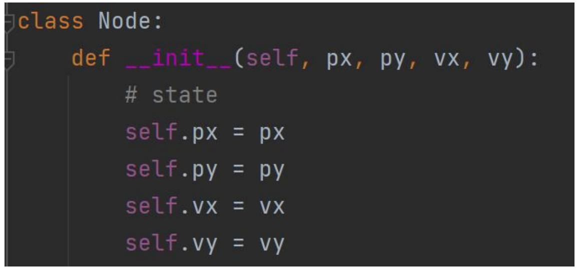
```
FINISH = 3
START = 2
FREE = 0
OCCUPIED = 1
OUTBOUND = -1
```

```
START_LINE = [[0, 3], [0, 4], [0, 5], [0, 6]]
```

Grid Map

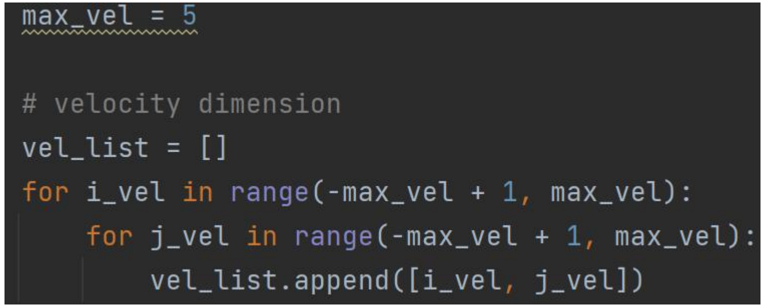
# 图的构建 － 节点

- state space



```python
class Node:
    def __init__(self, px, py, vx, vy):
        # state
        self.px = px
        self.py = py
        self.vx = vx
        self.vy = vy
```

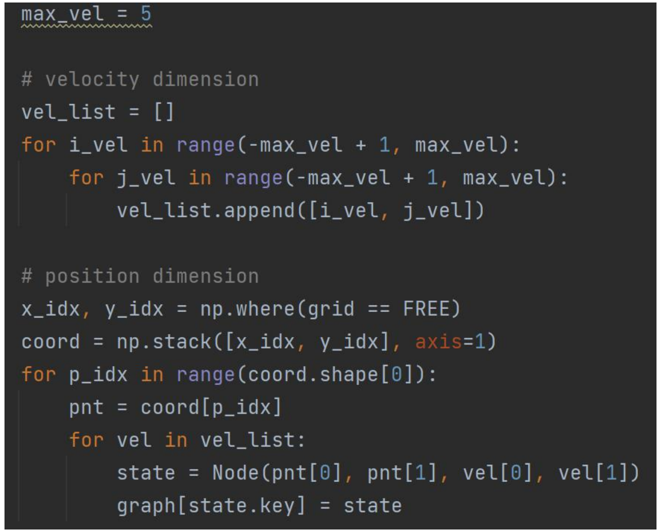
- 由于考虑了最后要停下来，这里还将速度也包含在state space中
- 所以每个 node 包含了 px py vx vy 四个状态
- 代码中速度的最大值设置为 4， 所以可能的速度为 [−4, −3, ⋯, 3, 4]



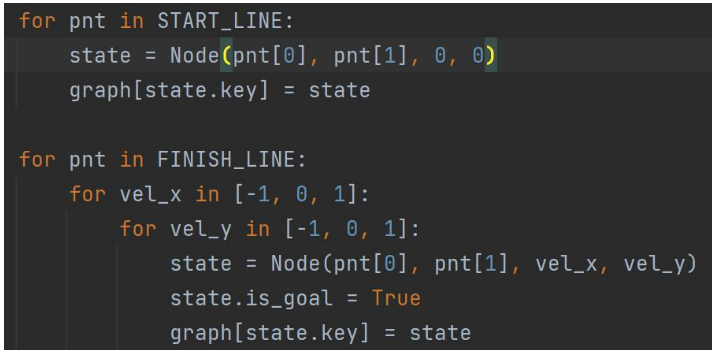
```python
max_vel = 5

# velocity dimension
vel_list = []
for i_vel in range(-max_vel + 1, max_vel):
    for j_vel in range(-max_vel + 1, max_vel):
        vel_list.append([i_vel, j_vel])
```

## 图的构建 – 节点

- 对于 START_LINE 和 FINISH_LINE 格子比较特殊
  - START_LINE vx = vy = 0
  - FINISH_LINE vx, vy 可能为 [-1, 0, 1] 这样下个时间才不会冲出去



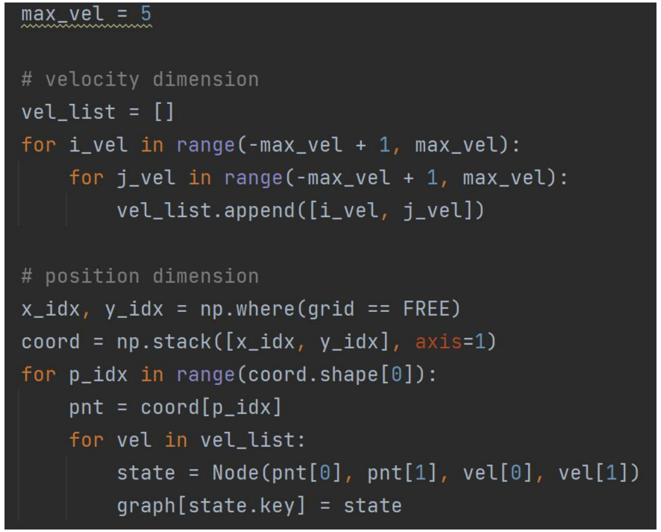
```python
max_vel = 5

# velocity dimension
vel_list = []
for i_vel in range(-max_vel + 1, max_vel):
    for j_vel in range(-max_vel + 1, max_vel):
        vel_list.append([i_vel, j_vel])

# position dimension
x_idx, y_idx = np.where(grid == FREE)
coord = np.stack([x_idx, y_idx], axis=1)
for p_idx in range(coord.shape[0]):
    pnt = coord[p_idx]
    for vel in vel_list:
        state = Node(pnt[0], pnt[1], vel[0], vel[1])
        graph[state.key] = state
```



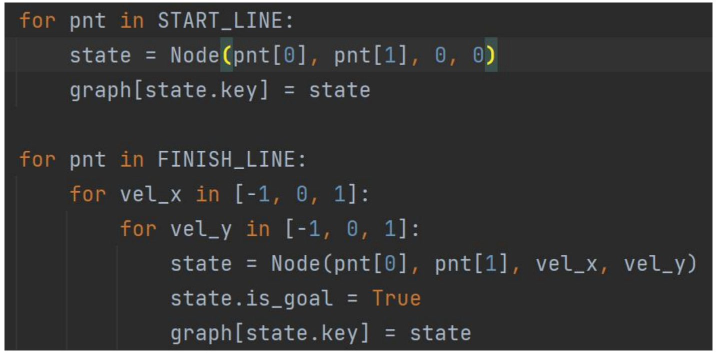
```python
for pnt in START_LINE:
    state = Node(pnt[0], pnt[1], 0, 0)
    graph[state.key] = state

for pnt in FINISH_LINE:
    for vel_x in [-1, 0, 1]:
        for vel_y in [-1, 0, 1]:
            state = Node(pnt[0], pnt[1], vel_x, vel_y)
            state.is_goal = True
            graph[state.key] = state
```

# 图的构建 － 边

- 对于 START_LINE 和 FINISH_LINE 格子比较特殊
  - START_LINE vx = vy = 0
  - FINISH_LINE vx, vy 可能为 [-1, 0, 1] 这样下个时间才不会冲出去



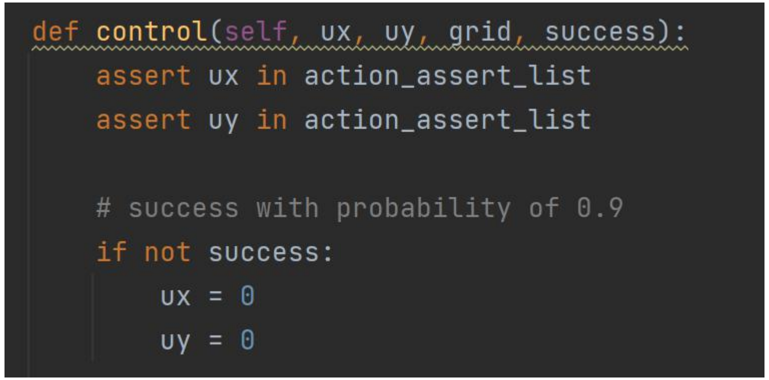
```python
max_vel = 5

# velocity dimension
vel_list = []
for i_vel in range(-max_vel + 1, max_vel):
    for j_vel in range(-max_vel + 1, max_vel):
        vel_list.append([i_vel, j_vel])

# position dimension
x_idx, y_idx = np.where(grid == FREE)
coord = np.stack([x_idx, y_idx], axis=1)
for p_idx in range(coord.shape[0]):
    pnt = coord[p_idx]
    for vel in vel_list:
        state = Node(pnt[0], pnt[1], vel[0], vel[1])
        graph[state.key] = state
```



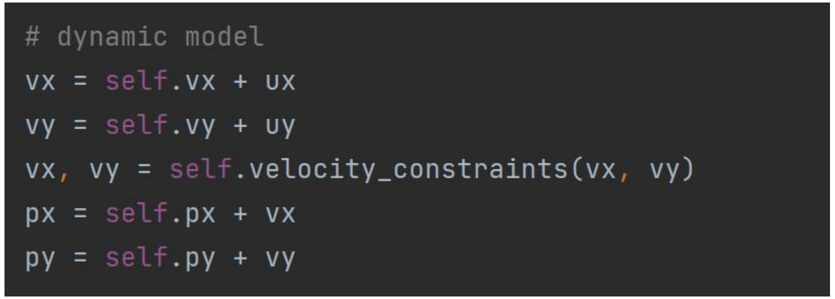
```python
for pnt in START_LINE:
    state = Node(pnt[0], pnt[1], 0, 0)
    graph[state.key] = state

for pnt in FINISH_LINE:
    for vel_x in [-1, 0, 1]:
        for vel_y in [-1, 0, 1]:
            state = Node(pnt[0], pnt[1], vel_x, vel_y)
            state.is_goal = True
            graph[state.key] = state
```

# 图的构建 － 边

- 这里先分析一下 action space
  - 我们的action为加速度， 取值为 [-1, 0, 1]
  - 通过control方法，我们可以计算出一个节点的后继节点有哪些
  - 有0.1的概率action都取0



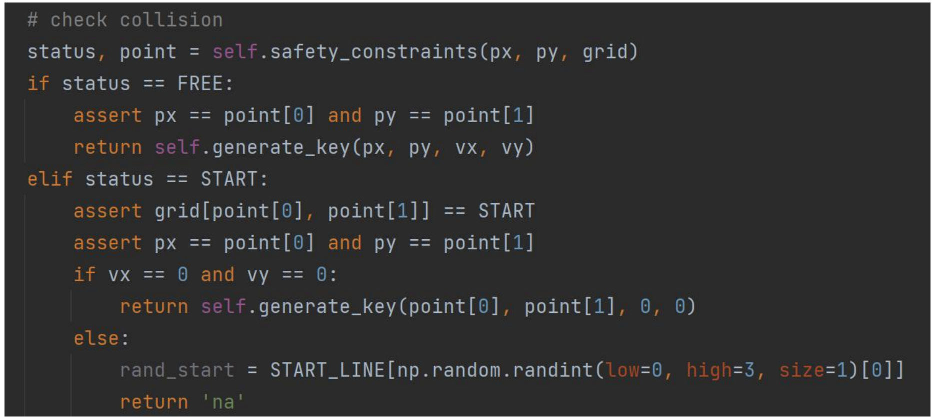
```
def control(self, ux, uy, grid, success):
    assert ux in action_assert_list
    assert uy in action_assert_list

    # success with probability of 0.9
    if not success:
        ux = 0
        uy = 0
```



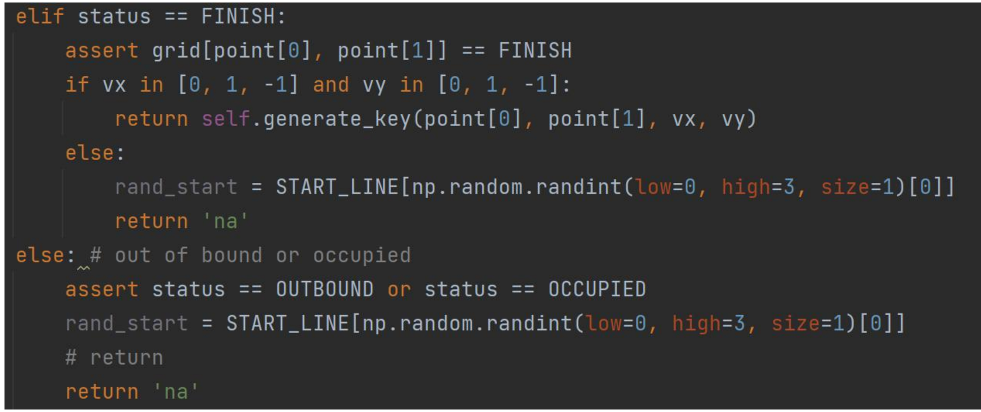
```
# dynamic model
vx = self.vx + ux
vy = self.vy + uy
vx, vy = self.velocity_constraints(vx, vy)
px = self.px + vx
py = self.py + vy
```

# 图的构建 － 边

- 有一些计算出的后继节点并不存在，可能是速度不符合，或位置碰到障碍或出界， 我们标记为‘na’



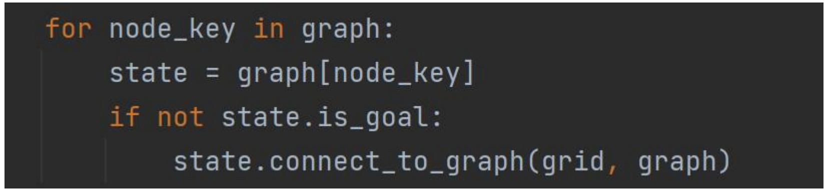
```
# check collision
status, point = self.safety_constraints(px, py, grid)
if status == FREE:
    assert px == point[0] and py == point[1]
    return self.generate_key(px, py, vx, vy)
elif status == START:
    assert grid[point[0], point[1]] == START
    assert px == point[0] and py == point[1]
    if vx == 0 and vy == 0:
        return self.generate_key(point[0], point[1], 0, 0)
    else:
        rand_start = START_LINE[np.random.randint(low=0, high=3, size=1)[0]]
        return 'na'
```



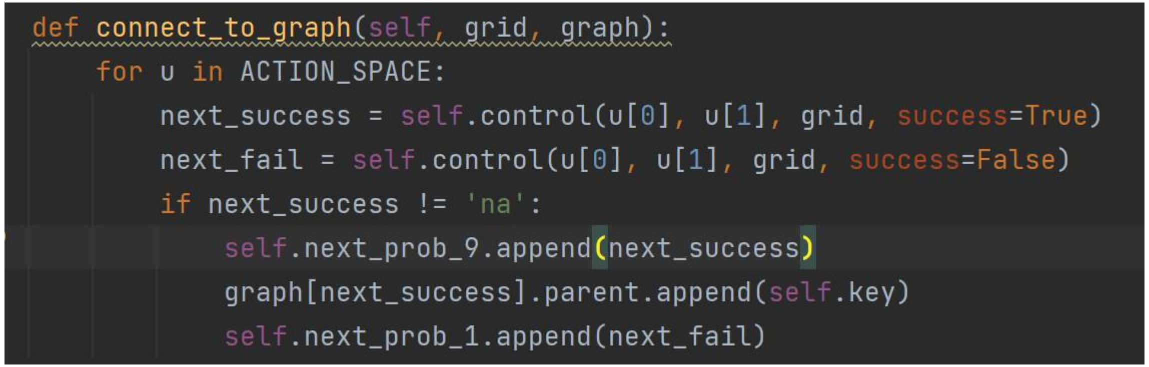
```
elif status == FINISH:
    assert grid[point[0], point[1]] == FINISH
    if vx in [0, 1, -1] and vy in [0, 1, -1]:
        return self.generate_key(point[0], point[1], vx, vy)
    else:
        rand_start = START_LINE[np.random.randint(low=0, high=3, size=1)[0]]
        return 'na'
else: # out of bound or occupied
    assert status == OUTBOUND or status == OCCUPIED
    rand_start = START_LINE[np.random.randint(low=0, high=3, size=1)[0]]
    # return
    return 'na'
```

# 图的构建 － 边

- 我们把这些边链接起来，　不存在的节点可以忽略



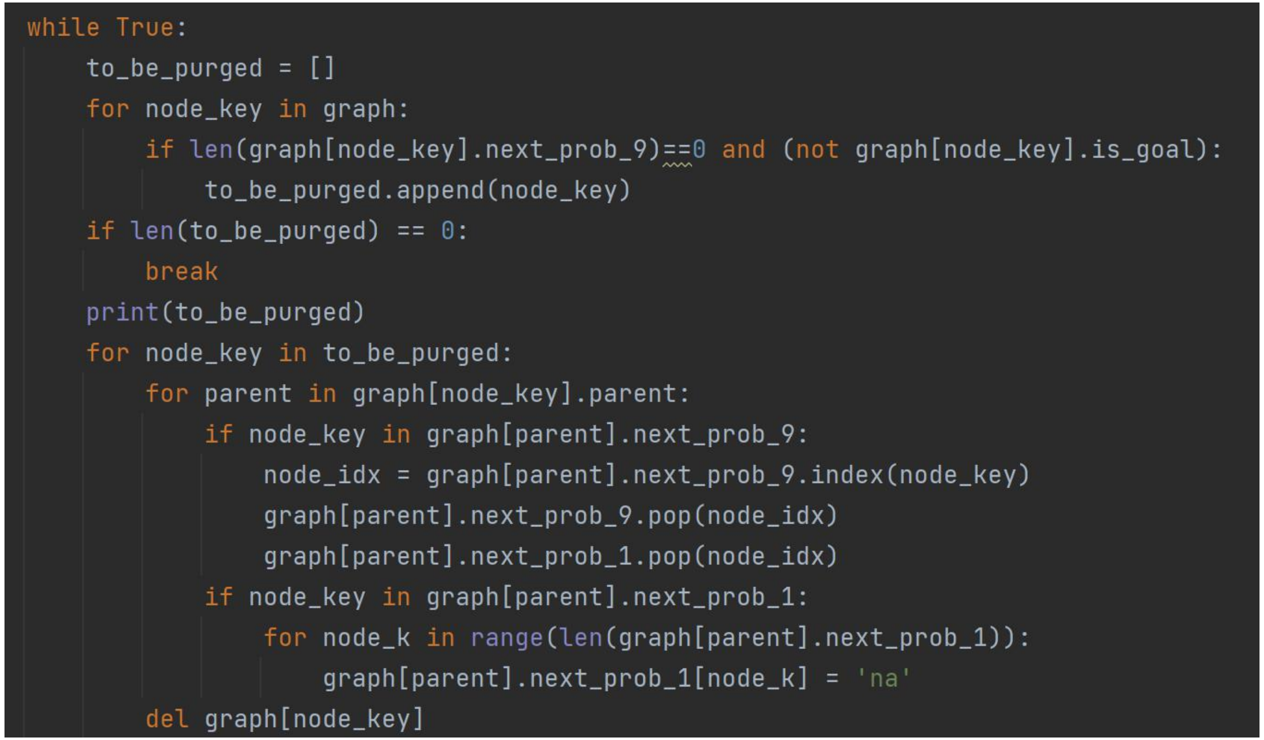
```python
for node_key in graph:
    state = graph[node_key]
    if not state.is_goal:
        state.connect_to_graph(grid, graph)
```



```python
def connect_to_graph(self, grid, graph):
    for u in ACTION_SPACE:
        next_success = self.control(u[0], u[1], grid, success=True)
        next_fail = self.control(u[0], u[1], grid, success=False)
        if next_success != 'na':
            self.next_prob_9.append(next_success)
            graph[next_success].parent.append(self.key)
            self.next_prob_1.append(next_fail)
```

## 图的构建 － 边

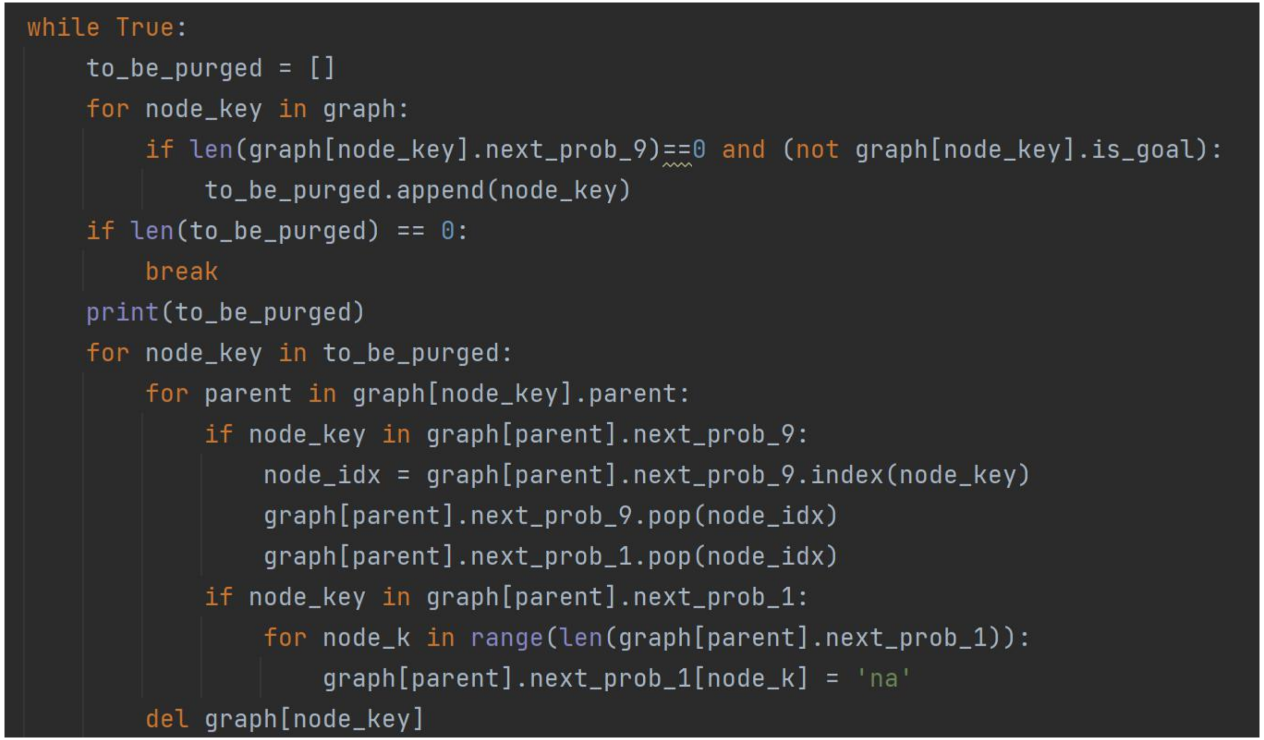
- 剪枝，　把没有后继节点的节点删掉　（end格子除外）



```
while True:
    to_be_purged = []
    for node_key in graph:
        if len(graph[node_key].next_prob_9)==0 and (not graph[node_key].is_goal):
            to_be_purged.append(node_key)
    if len(to_be_purged) == 0:
        break
    print(to_be_purged)
    for node_key in to_be_purged:
        for parent in graph[node_key].parent:
            if node_key in graph[parent].next_prob_9:
                node_idx = graph[parent].next_prob_9.index(node_key)
                graph[parent].next_prob_9.pop(node_idx)
                graph[parent].next_prob_1.pop(node_idx)
            if node_key in graph[parent].next_prob_1:
                for node_k in range(len(graph[parent].next_prob_1)):
                    graph[parent].next_prob_1[node_k] = 'na'
        del graph[node_key]
```

## 图的构建 － 边

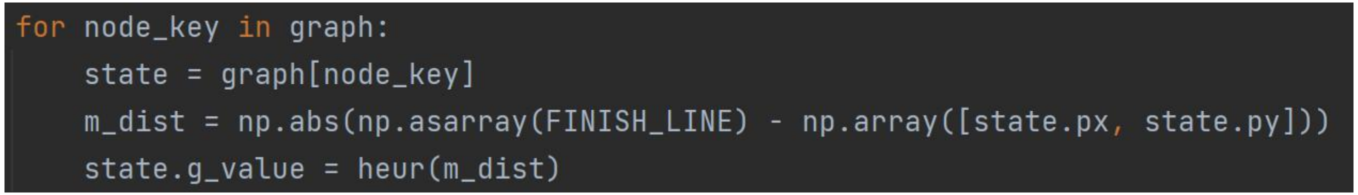
- 剪枝， 把没有后继节点的节点删掉



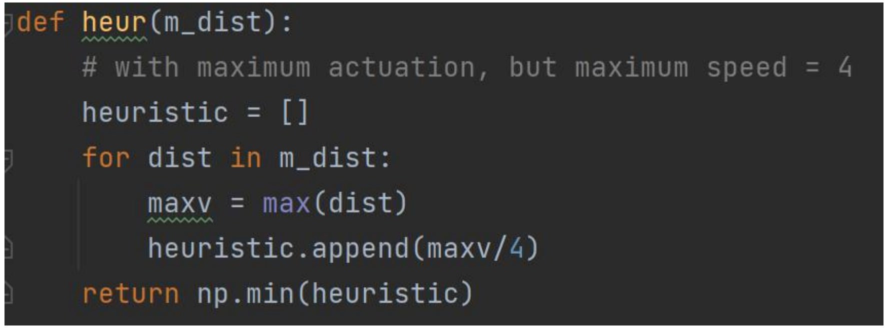
```python
while True:
    to_be_purged = []
    for node_key in graph:
        if len(graph[node_key].next_prob_9)==0 and (not graph[node_key].is_goal):
            to_be_purged.append(node_key)
    if len(to_be_purged) == 0:
        break
    print(to_be_purged)
    for node_key in to_be_purged:
        for parent in graph[node_key].parent:
            if node_key in graph[parent].next_prob_9:
                node_idx = graph[parent].next_prob_9.index(node_key)
                graph[parent].next_prob_9.pop(node_idx)
                graph[parent].next_prob_1.pop(node_idx)
            if node_key in graph[parent].next_prob_1:
                for node_k in range(len(graph[parent].next_prob_1)):
                    graph[parent].next_prob_1[node_k] = 'na'
        del graph[node_key]
```

# RTDP-初始化g_value

●



```python
for node_key in graph:
    state = graph[node_key]
    m_dist = np.abs(np.asarray(FINISH_LINE) - np.array([state.px, state.py]))
    state.g_value = heur(m_dist)
```



```python
def heur(m_dist):
    # with maximum actuation, but maximum speed = 4
    heuristic = []
    for dist in m_dist:
        maxv = max(dist)
        heuristic.append(maxv/4)
    return np.min(heuristic)
```

## 在线问答

Q&A

深蓝学院
shenlanxueyuan.com

# 感谢各位聆听！

Thanks for Listening