**In current setting we do not consider agent "turning" towards the direction which it chooses to move. Based on the previous assignment how would you propose to fix this short-coming in the context of the search problem we are solving?**

In the execute\_action we can fix this short-coming by changing the direction of the agent based on action given. Please have a look at the code below for elaboration

A picture containing text, screenshot, font

Description automatically generated

In the code above if the action is ‘UP’ then there can be 4 possible scenarios

* If the agent is facing left then we need to turn it to right direction before moving up
* If the agent is facing right then we need to turn left before we can move up.
* If the agent is facing down then we need to take left turn 2 time to face the agent in the upwards direction.
* If the agent is facing up then we can just go forward in that direction.

A similar code can be used for ‘DOWN’ ‘LEFT’ and ‘RIGHT’ direction. This is how we can fix the shortcoming.

**How does your proposed solution affect the branching factor and the number of nodes being added to the frontier at each step?**

'''  
if action == 'UP':  
 if agent.direction.direction == 'left':  
 agent.direction += Direction.R  
 agent.performance -= 1  
 elif agent.direction.direction == 'right':  
 agent.direction += Direction.L  
 agent.performance -= 1  
 elif agent.dirbction.direction == 'down':  
 agent.direction += Direction.L  
 agent.direction += Direction.L  
 agent.performance -= 2  
 agent.bump = self.move\_to(agent, agent.direction.move\_forward(agent.location))  
 if not agent.bump:  
 self.buttons[yi][xi].config(text="")  
 xf, yf = agent.location  
 self.buttons[yf][xf].config(text=agent\_label(agent))  
 agent.performance -= 1  
   
elif action=='DOWN':  
 if agent.direction.direction=='up':  
 agent.direction += Direction.R  
 agent.direction += Direction.R  
 agent.performance -= 2  
 elif agent.direction.direction=='left':  
 agent.direction += Direction.L  
 agent.performance -= 1  
 elif agent.direction.direction == 'right':  
 agent.direction += Direction.R  
 agent.performance -= 1  
 agent.bump = self.move\_to(agent, agent.direction.move\_forward(agent.location))  
 if not agent.bump:  
 self.buttons[yi][xi].config(text="")  
 xf, yf = agent.location  
 self.buttons[yf][xf].config(text=agent\_label(agent))  
 agent.performance -= 1  
 '''

In reference to the pseudo code given above for execute\_action function the branching factor will not change because we are changing the direction and moving the agent in the same step. This will not change the branching factor as we are still in that same node.

**How do you propose to consider turning effort to be added to the cost?**

Everytime agent does rotation (left, right, up and down) then we account for it and then add it to the cost. For example if action that agent is supposed to take is ‘UP’ and Agent is facing ‘Down’ then increment the cost by +2 and +1 if agent is facing left, right or up. Similarly, example if action that agent is supposed to take is ‘DOWN’ and Agent is facing ‘up’ then increment the cost by +2 and +1 if agent is facing left, right or down. Similar steps can take to account for cost in case the agent is supposed to take the action of ‘LEFT’ or ‘RIGHT’. Please have a look at the pseudo code below for elaboration

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

In the code pt will be the additional cost that we need to add to our path cost if we account for rotation as well. The final path\_cost will be calculated after adding ‘pt’ to the path\_cost.

**How do you think your strategy will affect the chosen path at each search?**

A screenshot of a keyboard

Description automatically generated with low confidence

In the Scanerio above, the agent is facing ‘UP’ and there is dirt present in ‘UP’ direction as well as ‘RIGHT’ direction. The 2 dirt location are 2 steps away from the agent current location on the grid shown above. However, if agent plans to clean the dirt on the right then cost will be higher because the agent first has to rotate right in order to face the dirt and then take 2 steps to reach the dirt location. In this case the total path cost will be 3 but if the agent plans to clean the dirt in the ‘UP’ then the cost will be less because the agent is already facing upwards and just has to take 2 steps to reach the dirt location without performing any rotation. In this case, the total dirt cost will be 2.

Conclusion:

The agent that accounts rotation in its path cost should go and clean the dirt in the ‘UP’ direction because the path cost will be less in this case.

Finally come up with a pseudocode describing how to implement this at search and at the visual rendering part.

'''  
agt=self.agent.location  
pt=0  
if (action=='UP'):  
 if self.agent.direction.direction=='right':  
 pt=pt+1   
 elif self.agent.direction=='left':  
 pt=pt+1  
 elif self.agent.direction=='down':  
 pt=pt+2  
 else:  
 pt=pt+0  
if (action=='DOWN'):  
 if self.agent.direction.direction == 'right':  
 pt = pt + 1  
 elif self.agent.direction == 'left':  
 pt = pt + 1  
 elif self.agent.direction == 'up':  
 pt = pt + 2  
 else:  
 pt = pt + 0  
  
if (action=='LEFT'):  
 if self.agent.direction.direction == 'right':  
 pt = pt + 2  
 elif self.agent.direction == 'down':  
 pt = pt + 1  
 elif self.agent.direction == 'up':  
 pt = pt + 1  
 else:  
 pt = pt + 0  
  
if (action=='right'):  
 if self.agent.direction.direction == 'left':  
 pt = pt + 2  
 elif self.agent.direction == 'down':  
 pt = pt + 1  
 elif self.agent.direction == 'up':  
 pt = pt + 1  
 else:  
 pt = pt + 0  
  
delta\_x = state1[0] - state2[0]  
delta\_y = state1[1] - state2[1]  
z1 = np.sqrt(state1[0] \* state1[0] + state1[1] \* state1[1]) # state1  
z2 = np.sqrt(state2[0] \* state2[0] + state2[1] \* state2[1]) # state2  
distance = (delta\_x \*\* 2 + delta\_y \*\* 2 + (z1 - z2) \* (z1 - z2))  
cost = c + distance +pt  
return cost  
'''

In the pseudo code above ‘pt ‘ is the additional cost that will be take into account when considering turning/rotation. The final path cost will account for ‘pt’ as well.