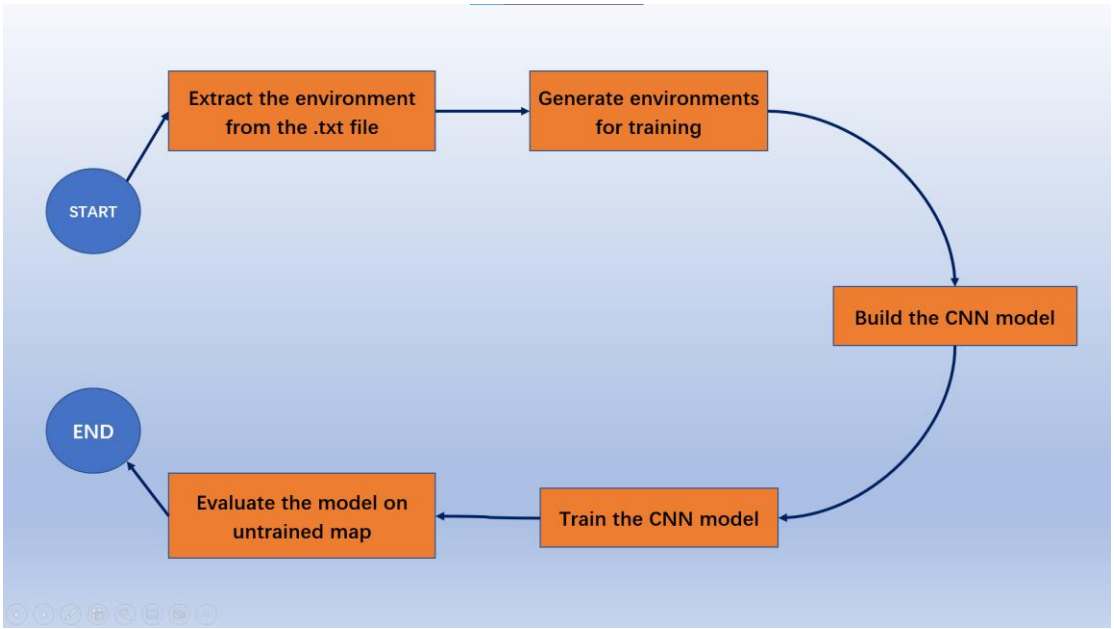
Deep Reinforcement Learning on finding the shortest path

Overall flowchart



A. Extract the environment from the .txt file

- To extract the environment from the text file, we will need to pass the filename to the Environment class.
- The Environment class have the following functions:

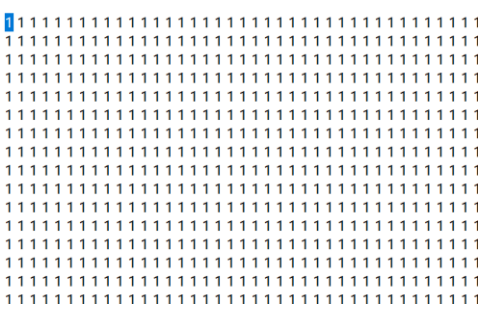
No	Function	Description
1	transform_env(filename)	return the environment (numpy 2D) in number form given the file in symbol form (.SWT)
2	print_properties()	print out the properties of the environment
3	va_move(R,C)	return the valid movement (numpy 1D)
4	step(actions, update_step = True)	If update_step = True, it Perform the action and return the obs, reward, done If update_step = False, it will not perform the action and only return the reward if perform that actions
	- get_obs(R,C)	return the obs at position R,C
	- get_reward(R, C, record_env, update_step = True)	return the reward after an action
	- isdone()	return whether the game is end or not
	- distance(x1, y1, x2, y2)	return the city block distance between (x1,y1) and (x2,y2)
5.	reset()	Reset the map
6.	obs_plot(actions, title, animation = True)	Plot the actions of the agents in a map

1. transform_env(filename)

- First, this function will transform the following terrain to number:

Num	Terrain
0	Obstacle 'W'
1	Land '.'
2	Starting point(S)
3	Target (T)
4	battle fog/unknown terrain

- Then, it will return the environment in NumPy 2D array that padding with 0 for 1 round & padding for 4 for 11 rounds.
- Therefore, since the original dimension = (400, 365), the output dimension will be (424, 389)
- The reason of doing so (padding) is to prevent not enough observations when we are at the corner of the map. For example:

	
With padding, we can get 25x25 observation	Without padding, we only will get 11x11 observation which will cause error when feed into the deep learning model.

** The reason of why is 25x25 obs but not 21x21 obs will be explained in (4)

2. print_properties()

- if the argument print_info = True, it will print out the properties of environment. For example:

```
For the file Maps/map1_trans.txt:
num_W = 2184
num_land = 145346
num_S = 1
num_T = 3
num_unknown = 17402
```

3. va_move(R, C)

- Since the agent can move in 8 directions, these movement is characterized by 8 different number (from 0 to 7):

Index	Move
0	Right
1	Right Up
2	Up
3	Left Up
4	Left
5	Left Down
6	Down
7	Right Down

- When we are location R, C, we will check for those 8 directions,

  i) For Right, Up, Left, Down,

  - if index *i* not in obstacle, vaMove[i] = 1

```
# Check for Right(0), Up(2), Left(4), Down(6),
for i,(x,y) in enumerate(((0,1),(-1,0),(0,-1),(1,0))):
    if self.env[R+x, C+y] not in OBSTACLE:  # If not in obstacle, it is a valid move
        vaMove[2*i] = 1
```

  ii) For Right Up, Left Up, Left Down, Right Down, it is a bit different. For example, if Right Up and 'Right' and 'Up' not in obstacle, then we can set 'Right Up' is valid move.

```
# Check for Right Up(1), Left Up(3), Left Down(5), Right Down(7)
for i,(x,y) in enumerate(((-1,1), (-1,-1), (1, -1), (1,1))):
    if vaMove[(2*i)] and vaMove[(2*i+2)%8] and self.env[R+x,C+y] not in OBSTACLE:
        vaMove[2*i+1] = 1
```

  - This for loop is equivalent to the 4 if … at below:

```
# Check for Right Up(1)
if vaMove[0] and vaMove[2] and env[R-1,C+1] not in OBSTACLE:
    vaMove[1] = 1

# Check for Left Up (3)
if vaMove[2] and vaMove[4] and env[R-1, C-1] not in OBSTACLE:
    vaMove[3] = 1

# Check for Left Down (5)
if vaMove[4] and vaMove[6] and env[R+1, C-1] not in OBSTACLE:
    vaMove[5] = 1

# Check for Right Down (7)
if vaMove[6] and vaMove[0] and env[R+1, C+1] not in OBSTACLE:
    vaMove[7] = 1
```

iii) After that, we have made some effort on make sure the agent will not go through the previous position that have been pass through:

```
vaMoveCount = np.full(8,1000, dtype = np.int16)
for action in range(vaMove.shape[0]):   # action = [0, 1, 2, 3, 4, 5, 6, 7]
    if vaMove[action] == 1:             # If it is a valid move
        move = self.move_dic[action]
        if self.recorded_RC[R + move[0], C + move[1]] > 0:  # If the agent already go through it,  it wont go though again
            vaMoveCount[action] =  self.recorded_RC[R + move[0], C + move[1]]
            vaMove[action] = 0
```

- Note that at the __init__(…), we have set that all the obstacles already been passed through 1000 times so that it will never go through the obstacles:

```
self.recorded_RC = np.zeros((424,389), dtype = np.int16)   # Record R, C to minimize the action that go back to original places
self.recorded_RC[self.curr_R, self.curr_C] = 1
self.recorded_RC[np.where(self.env == self.W)] += 1000      # Set all the obstacle already pass through 1000 times
```

iv) Then, if all its surround already goes through for one time, make it only can go through the one with least number of pass through

```
if vaMove.sum() == 0:
    vaMove = vaMoveCount == vaMoveCount.min()
```

- At last, this function will return a 1D array with size (8,). For example:

  [1, 1, 1, 1, 1, 0, 1, 0] implies that left down (5) and right down (7) is not a valid movement, while other movement is valid move.

4. step(actions, update_step = True)

- At the beginning, this function will check where update_step is True or False.
  - If update_step = True, it Perform the action and return the observation, reward, done

- If update_step = False, it will not perform the action and only return the reward if perform that action

- We will focus on when update_step = True as when update_step = False, we just cancel out some part of the coding.

- First, it will update the position (R,C) based on the action (range from 0 – 7) that described in the figure at 3.va_move(R,C). We have used a dictionary to update the step by following:

```
self.move_dic = {0: (0,1), 2:(-1,0) , 4:(0,-1), 6:(1,0),
                 1:(-1,1), 3:(-1,-1), 5:(1, -1), 7:(1,1)}
```

```
# 1. Update the current position
self.curr_R, self.curr_C = self.curr_R + move[0], self.curr_C + move[1]
```

- Then, we will get the new observation after performing the step and record it:

```
# 2. Get the new observation at a given position and record it
obs = self.get_obs(self.curr_R, self.curr_C)
self.recorded_env[self.curr_R-10: self.curr_R+11, self.curr_C-10: self.curr_C+11] = np.copy(obs)
```

- After that, we will remove the target in 'self.target' if found it and get whether the game is end or not by calling function 'self.isdone()':

```
# 3. Remove the target if found it and Get whether the game is end or not
for i, (T_R, T_C) in enumerate(self.target):
    if self.curr_R == T_R and self.curr_C == T_C:
        true_false = np.full(self.target.shape[0],True)
        true_false[i] = False
        self.target = self.target[true_false]                    # Remove the target if we find it
        self.env[T_R, T_C] = self.LAND                           # Set the target as land
        self.recorded_env[T_R, T_C] = self.LAND                  # Set the target as land
        found_T = True
        break
done = self.isdone()
```

- Next, we will get the reward, if we reach target, we will add 10 reward, else, we will use get_reward() function to compute the reward. When we use get_reward() function, we will decrease the reward and increase the penalty when time go by and when we move in direction (Right Up, Left Up, Left Down, Right Down). It is because moving in (Right Up - 1, Left Up - 3, Left Down - 5, Right Down - 7) cost 14, but moving in (Right - 0, Left - 2, Up - 4, Down - 6) only cost 10.

```
div_cost = [1, 1.1]
```

```
# 4. Get the reward after performing the action
if found_T:
    r = 10.0
else:   #  (reward by replacing UKN(4) with other value)
    r = self.get_reward(self.curr_R, self.curr_C, self.recorded_env)

    """ When time go by, the reward will continue decreasing,
        When move Left/Right/Up/Down the ----------------------> reward is remain
             move Right Up/Left Up/Left Down(5)/Right Down(7)--> reward is divide by 1.1 """
    if r > 0:
        r = r * (self.gamma ** (self.time_step// 100)) / div_cost[action % 2]  # Decrease the reward

    else:
        r = r / (self.gamma ** (self.time_step// 100)) * div_cost[action % 2]  # Increase the penalty
```

- Lastly, we will update the time step & cost, accumulate the reward and return the 25x25 recorded observations, reward and done:

```
# 5. Update the time_step
self.time_step += 1
self.total_cost += cost[action % 2]
self.curr_reward += r

self.recorded_RC[self.curr_R, self.curr_C] += 1

# return an 25x25 observation while may contain unknown terrain, r, done
return self.recorded_RC[self.curr_R-12: self.curr_R+13, self.curr_C-12: self.curr_C+13], \
       self.recorded_env[self.curr_R-12: self.curr_R+13, self.curr_C-12: self.curr_C+13], self.curr_reward, done
```

- The observation is 25x25 but not 21x21 can be explained by the figure below:

Original observation (21x21)	Our view (25x25)

*** After we observe the observation, we will record in in recorded_env, if we haven't observed the observation before, it will be fill by unknown(4).

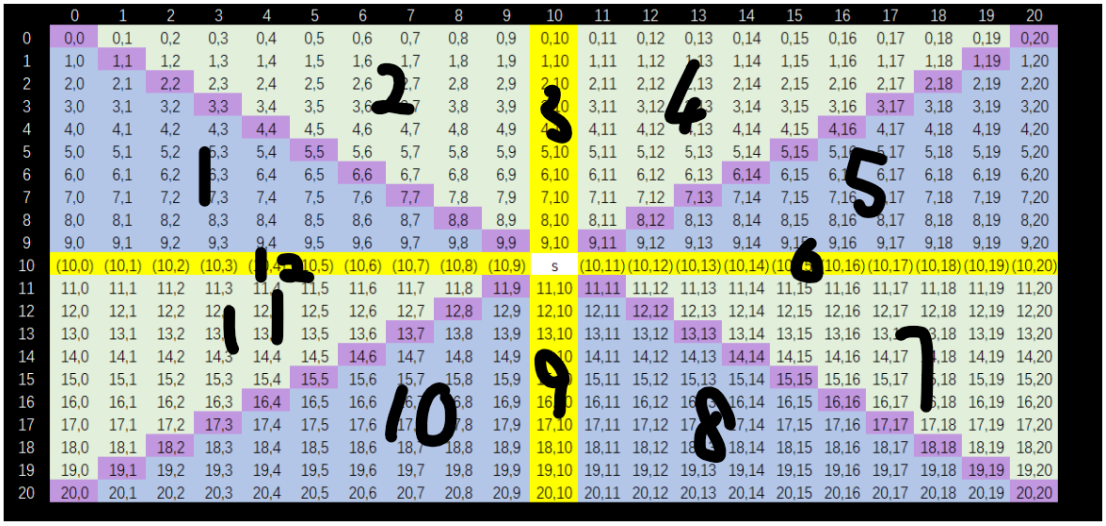
*** The benefit of doing this is that the model will know which direction haven't been explored so that it will favour the move that explore new observations

4.1 get_obs(R,C)

- This function will return 21x21 observations at a given position (R,C)
- The concept of getting the observation is as below:

i) First, the observation is divided into 12 part:

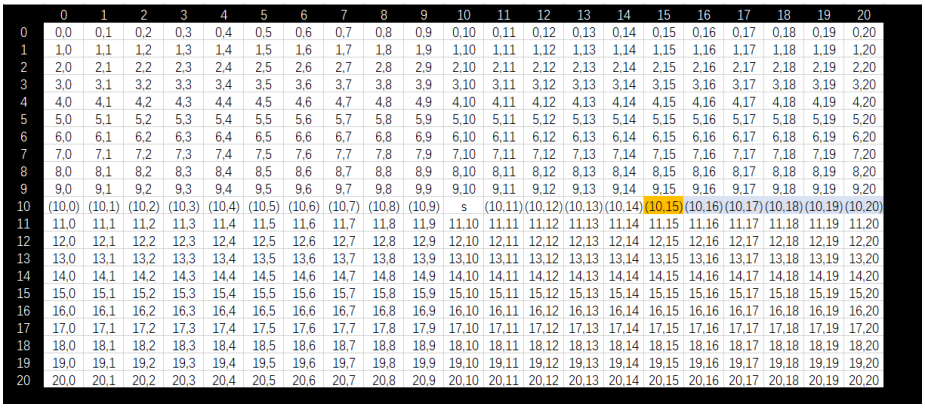
top(3), bottom(9), left(12), right(6)
top left up(2), top left down(1)
top right up(4), top right down(5)
botton left up(11), bottom left down(10)
botton right up(7), bottom right down(8)



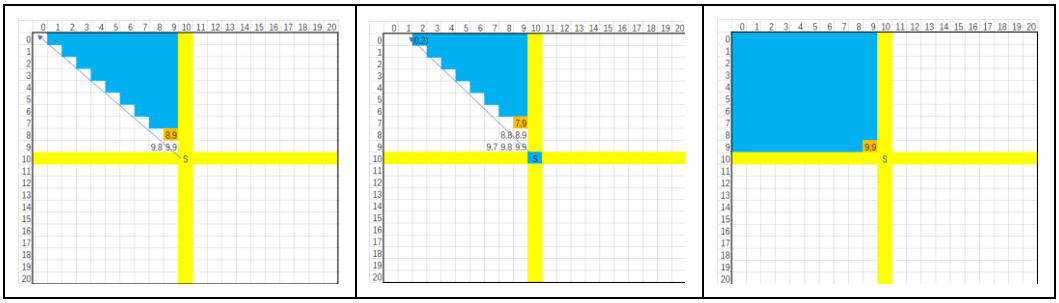
ii) Then, we will fill the terrain that suppose should not be observed to be the UNKNOWN terrain:

For top(3), bottom(9), left(12), right(6), if we found an obstacle, the terrain behind the obstacle will be set as unknown:

For example, if position (10,15) is obstacles, (10,16), (10,17), (10,18), (10,19), (10,20) will be set to unknown terrain which is represented by number 4.



iii) For the remaining part of the map, it only can observe the terrain that is in the view. For example, orange colour represents the obstacle, blue colour represent the unknown terrain, the arrow represent the agent's view



iv) Lastly, we will fill the unknown terrain with value that have been recorded before and return an 21x21 numpy array

```
# Fill the unknown with value that have been recorded before
cfo_UNK = curr_full_obs == self.UNK
curr_full_obs[cfo_UNK] = np.copy(self.recorded_env[R-10: R+11, C-10: C+11][cfo_UNK])
return curr_full_obs
```

4.2 get_reward(R, C, record_env, update_step = True)

- for getting the reward, we have set

  2 rewards

  - reward for observing new observation
  - reward for going near to the target once the agent see the target

  3 penalties

  - penalty for not observing new observations
  - penalty for away from the target once the agent see the target
  - Penalty for going back to position that have moved before

- Note that the penalty for away from the target is more than the reward for going near the target so that it will not keep loop around the target
- Lastly, this function will return the reward in range [-3.6, 1.805]

4.3 isdone() – This function return Boolean value on whether the game is end or not by checking whether there is still have target in the map

```
def isdone(self):
    if self.target.shape[0] == 0:
        return True
    return False
```

4.4 distance(x1, y1, x2, y2) – return the city block distance

- This function is used in the get_reward() function to compute the city block distance between agent and target

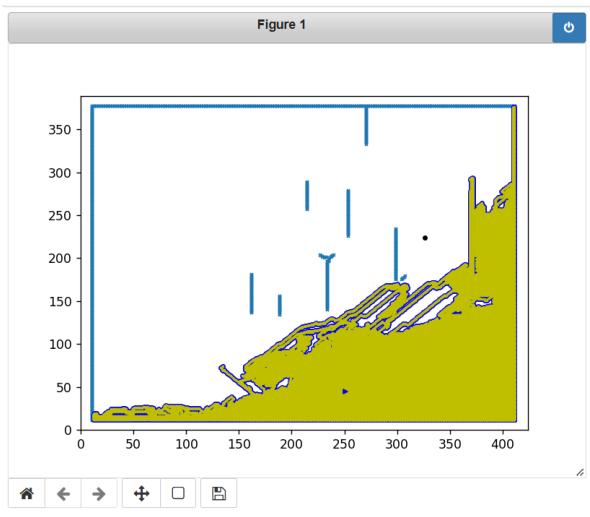
```
def distance(self, x1, y1, x2, y2):
    return np.abs(x1 - x2)  + np.abs(y1 - y2)
```

5. reset() – this function will reset the environment by calling the __init__ again:

```
def reset(self):
    self.__init__(self.filename, False)
```

6. obs_plot(actions, title animation = True)

- Given an array of actions in range [0,7] and the title of the graph, this function will show the movement of the agent. For example:

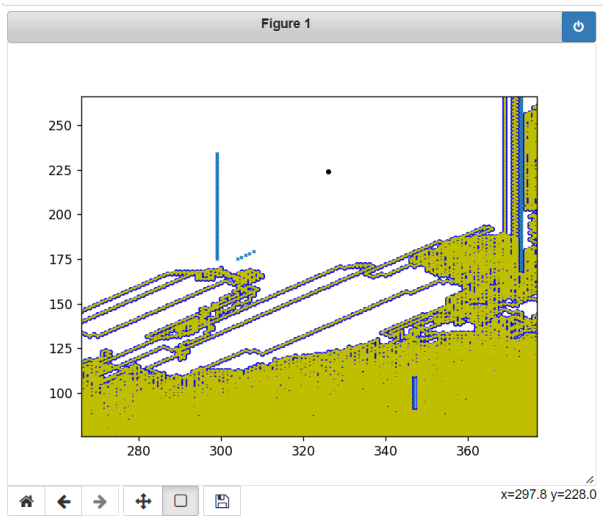


The blue arrow is the current position,

The black dot is the target

The green on is the path

We can zoom in the map by selecting ▢

	Example of zooming in a map. The blue cross ✖ is obstacle

- If want to see the animation of the code, need to use .py file and set the argument animation = True. However, it is not suggested as it will take a very long time for an agent to complete a map

B. Generate environments for training

Since we are developing deep neural networks, we will need more data for training. Therefore, we have created a function "generate_env(filename)" to generate the environment with the following parameters:

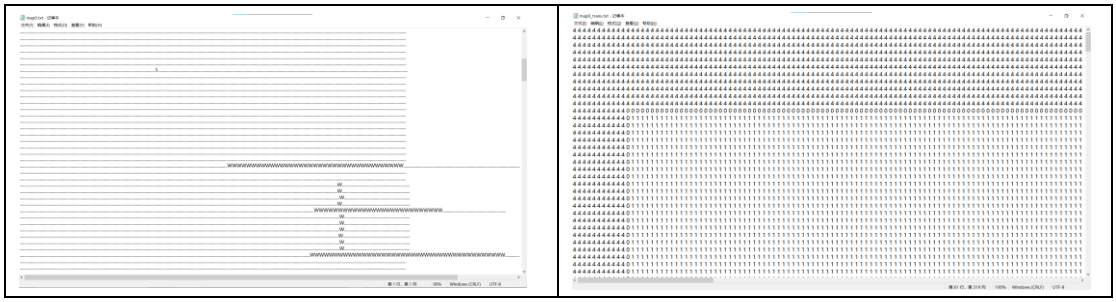
```
MIN_NUM_W_ROW = 20
MAX_NUM_W_ROW = 50
MIN_NUM_W_COL = 5
MAX_NUM_W_COL = 15
W = 0
LAND = 1
S = 2
T = 3
UNK = 4
```

While generating the environment, 2 files will be created at the "/Maps" directory:

- 1 file with number and padding (mapi_trans.txt)
- 1 file with "WST" notation and without padding (mapi.txt)

where i = 0, 1, 2, 3, 4, 5, …

Sample of mapi.txt	Sample of mapi_trans.txt



The reason of having a mapi_trans.txt is to reduce the time to convert map from “WS” notation to number with padding.

Additionally, the reason of having a mapi.txt is to easier the visualization of the map.

C. Build the CNN model

```
class CNN(nn.Module):
    def __init__(self):
        # call the parent constructor
        super(CNN, self).__init__()

        # Input shape = (batch_size, 6, 25, 25)
        self.layer1 = nn.Sequential(nn.Conv2d(6, 16, kernel_size=(5, 5), stride=1, padding='same'),
                                    nn.ReLU(),
                                    nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

        self.layer2 = nn.Sequential(nn.Conv2d(16, 8, kernel_size=(3, 3), stride=1, padding='same'),
                                    nn.ReLU(),
                                    nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

        self.layer3 = nn.Sequential(nn.Linear(in_features=288, out_features=64),
                                    nn.Dropout(p=0.3),
                                    nn.ReLU(),
                                    nn.Linear(in_features=64, out_features=8),
                                    nn.Softmax(dim=1))
```

Since the input is observation, we decided to use convolution layer as this layer can capture the information about the neighbours of a position. Note that before layer3, we need to flatten the feature to fix into Linear Layer that can only take 2D input:

```
def forward(self, x):
    x = self.layer1(x)          # Output shape = [2, 16, 12, 12]
    x = self.layer2(x)          # Output shape = [2, 8, 6, 6]
    x = torch.flatten(x, 1)     # Output shape = (2, 576 = 8*6*6)
    output = self.layer3(x)

    return output
```

In total, we have 22592 trainable parameters in CNN:

```
model = CNN()
total_params = sum(param.numel() for param in model.parameters())
print("Number of total parameter in CNN =",total_params)
```

```
Number of total parameter in CNN = 22592
```

D. Train the CNN model

To train the CNN model, we need several functions to help:

No	Function	Description
1	get_first_obs(Map)	Return first observation and valid movement of agent in the Map
2	get_next_obs(obs, RC_obs, Map)	Return next observation and valid movement of agent in the Map
3	get_action_n_prob(obs, va_move, Map, model, train_mode = True)	Given the observation, get the action of the agent and probability to perform the actions
4	train(Map, optimizer, batch_size, device, max_step = 2000, min_reward = -95.0, RDC = 1)	Train the model for 1 epoch

1. get_first_obs(Map)

- Firstly, this function will get a 25x25 observation from the recorded_env
- Then, we will use one-hot encoding on these observation as the terrain is represent from (0 to 4), so it is better to transform it into one-hot vector so that all of them have same contribution to the model
- After that, this function will get a 25x25 RC_obs to indicate that the number of passes through at a given position and concatenate with the 25x25 observation from the recorded_env
- Next, it will get the valid move of the agent at the current position
- Lastly, it will return the observation with shape (1,6,25,25) and valid movement with shape (1,8)

```
def get_first_obs(Map):
    obs = Map.recorded_env[Map.curr_R-12: Map.curr_R+13, Map.curr_C-12: Map.curr_C+13]                    # Shape = (25, 25)
    OH_obs = F.one_hot(torch.tensor(obs, dtype = torch.int64), num_classes=5)[None,:].permute(0,3,1,2)   # Shape = (1, 5, 25, 25)
    OH_obs = torch.as_tensor(OH_obs, dtype = torch.float32)
    RC_obs = torch.as_tensor(Map.recorded_RC[Map.curr_R-12: Map.curr_R+13, Map.curr_C-12: Map.curr_C+13][None,None,:,:],
                             dtype = torch.float32)                                                       # Shape =(1, 1, 25, 25)
    OH_RC_obs = torch.cat((OH_obs, RC_obs),1)                                                             # Shape =(1, 6, 25, 25)
    va_move = torch.as_tensor(Map.va_move(Map.curr_R, Map.curr_C)[None,:], dtype = torch.float32 )        # Shape =(1, 8)
    return OH_RC_obs, va_move
```

2. get_next_obs(obs, RC_obs, Map)

- This function is similar as the previous function.

- The only difference is that this function returns next observation and valid movement of agent in the Map

3. get_action_n_prob(obs, va_move, Map, model, train_mode = True)

- After we get the observation and valid move, we can get the action by feeding the observations into the model.
- The output of the model will be a probability distribution as the last layer used is softmax.

```
logits = model(obs)                                    # Logits for training the model
```

- After that, we will remove the invalid logits (by multiply with valid movement) and make the logits back to a probability distribution using softmax function on the logits that > 0.

```
logits_copy = logits.clone().detach()                  # Generate a copy of logits to generate the action
logits_copy = va_move * logits_copy                    # Remove the logits with invalid move
# Make the logits back to probability distribution to select an action
logits_copy[va_move != 0] = torch.exp(logits_copy[va_move != 0]) / torch.exp(logits_copy[va_move != 0]).sum()
```

- If train_mode = True, this function will return the target logits, logits, action that sampled from the logits OR action that sampled from maximum value
  - First, it will get all the expected rewards gotten after performing each valid step.

```
# Expected Reward to get after performing the valid movement
reward = np.full(8, -100.0, dtype = np.float64)
for action in range(np.array(va_move[0].to("cpu"),dtype = np.int16).shape[0]):
    if va_move[0][action] == 1:                # for each of the valid movement, calculate the reward gotten after perform the action
        reward[action] = Map.step(action, update_step = False)
```

  - After that, it is converted to the target logits, by following:

    If the target is found at the next move, the target probability to perform the step to reach target = 1, while the target probability of other movement will be 0

    If the target is not found, we will first normalized the reward to 0-1, and then the target logits will be calculated using the formula similar to softmax but with power of 10.

Softmax	Our function
$\frac{e^{z_i}}{\sum e^{z_i}}$	$\frac{e^{10z_i}}{\sum e^{10z_i}}$

```
# Get the target_logits
if torch.max(reward) == 10:    # If the target is found
    target_logits = torch.zeros(1,8, dtype = np.float64, device = device)
    target_logits[reward == 10] = 1
    target_logits = target_logits[None,:]
else:
    # 1. Normalize the reward to 0 - 1
    reward = (reward - reward.min()) / (reward.max() - reward.min())
    target_logits = reward.clone()

    # 2. Generate the probability distribution based on softmax function on the valid move ONLY!
    exp_reward = torch.exp(reward[reward !=0]) ** 10    # exponential reward for the valid movement
    sum_exp = exp_reward.sum()
    target_logits[reward != 0] = exp_reward/sum_exp
    target_logits = target_logits.view(1,8)
```

** The reason of normalize to 0-1 is to make the lowest reward to be 0 and the target logits on that movement will be 0. Initially, if there exist invalid movement in one of the direction, the lowest reward will be -100

```
reward = np.full(8, -100.0, dtype = np.float64)
```

** The reason of doing power of 10 is to make the agent more favor the movement with more reward and disfavor the movement with less reward. For example:

```
reward = torch.rand(1,8)
target_logits = reward.clone()

print("Reward gotten =",reward,"\n")
for i in range(1,11):
    exp_reward = torch.exp(reward[reward !=0]) ** i    # exponential reward for the valid movement
    sum_exp = exp_reward.sum()
    target_logits[reward != 0] = exp_reward/sum_exp
    print("When power =",i,"target logits =", target_logits)
```

```
Reward gotten = tensor([[0.0314, 0.2492, 0.1079, 0.3560, 0.2526, 0.1288, 0.6582, 0.6798]])

When power = 1 target logits = tensor([[0.0923, 0.1147, 0.0996, 0.1276, 0.1151, 0.1017, 0.1727, 0.1764]])
When power = 2 target logits = tensor([[0.0643, 0.0994, 0.0749, 0.1230, 0.1001, 0.0781, 0.2252, 0.2351]])
When power = 3 target logits = tensor([[0.0423, 0.0812, 0.0532, 0.1119, 0.0821, 0.0566, 0.2771, 0.2956]])
When power = 4 target logits = tensor([[0.0263, 0.0630, 0.0358, 0.0965, 0.0638, 0.0389, 0.3233, 0.3524]])
When power = 5 target logits = tensor([[0.0157, 0.0467, 0.0230, 0.0795, 0.0474, 0.0255, 0.3605, 0.4015]])
When power = 6 target logits = tensor([[0.0090, 0.0334, 0.0143, 0.0633, 0.0340, 0.0162, 0.3881, 0.4417]])
When power = 7 target logits = tensor([[0.0051, 0.0232, 0.0086, 0.0491, 0.0238, 0.0100, 0.4070, 0.4732]])
When power = 8 target logits = tensor([[0.0028, 0.0159, 0.0051, 0.0373, 0.0163, 0.0061, 0.4189, 0.4976]])
When power = 9 target logits = tensor([[0.0015, 0.0107, 0.0030, 0.0280, 0.0111, 0.0036, 0.4255, 0.5166]])
When power = 10 target logits = tensor([[0.0008, 0.0072, 0.0017, 0.0209, 0.0074, 0.0022, 0.4284, 0.5314]])
```

When power increasing, the target logits of the reward 0.6798 become higher.

- Lastly, this function will return the target_logits, logits and action, we can sampled the action by maximum value or sampled by probability. By based on our experiment, sampled by probability will help to train the model faster as it can get more difference observation on a same map.

```
        action = np.random.choice([0,1,2,3,4,5,6,7], p=logits_copy.view(-1).to("cpu").numpy())     # Random sample based on probability

        return target_logits, logits, action                            #  Train using sample by probability distribution
#         return target_logits, logits, torch.argmax(logits_copy).item()    # Train using sample by max
```

- If train_mode = False, this function will only return the action by selecting the actions with highest probability

```
else:
    return torch.argmax(logits_copy).item()
```

4. **train(Map, optimizer, batch_size, device, max_step = 2000, min_reward = -95.0, RDC = 1)**

- First, this function will get the first observation and valid move by calling get_first_obs(Map) function.
- Then, it will get an action and act on the environment

```
# Get an action and act in the environment
OH_RC_obs = OH_RC_obs.to(device)
va_move   = va_move.to(device)
target_p, p, action = get_action_n_prob(OH_RC_obs, va_move, Map, model)  # action is an integer in range [0, 7]
```

```
RC_obs, obs, reward, done = Map.step(action)
```

- If the game is end or the time step > max step allowed or the reward is continuous decreasing for 3 times or the reward is < min_reward allowed, it will reset the map.

  Else, it will get the next observation and valid move:

```
# Note that if these conditions is true, the action will not be used for update the weight of model
# if continuous decrease reward RDC times OR reward < min_reward allowed, reset the env without updating the bad action
if r_dec_count == RDC or reward < min_reward:
    if reward < min_reward:
        print(f"The reward is too low limit =",min_reward, "NOW:",reward)
    Map.reset()     # Reset the map
    prev_reward, r_dec_count = 0, 0
    OH_RC_obs, va_move = get_first_obs(Map)
    continue
```

```
# if episode is over OR time > max_step 0, reset the env
if done or Map.time_step >=  max_step:
    if done:
        print(f"> > > The Game end! with cost {Map.total_cost} ({Map.time_step} step)")
    Map.reset()     # Reset the map
    prev_reward, r_dec_count = 0, 0
    OH_RC_obs, va_move = get_first_obs(Map)
else:
    OH_RC_obs, va_move = get_next_obs(obs, RC_obs, Map)
```

- After that, the while loop will break until the batch_size is reach:

```
if batch_actions.shape[0] == batch_size :
    break
```

- Next, this function will compute the loss function by adding the cross-entropy loss and value loss with their corresponding coefficient (This coefficient is a hyperparameter which can be set by us)

```
# Calculate the cross entropy (update globally, we more trust cross entropy)
entropy_loss = - (batch_target_p * torch.log2(batch_p)).sum(1).mean()
entropy_coefficient = 1.1

# Calculate the value loss (if get + reward, more likely to move that move, if get -reward, less likely to move that move)
value_loss = - (torch.log2(batch_p)[ [[range(batch_actions.shape[0])], [batch_actions.view(-1).tolist()]] ] * \
               batch_rewards.view(-1)).mean()

# The coefficient will keep decreasing when time go as reward is accumulated
if Map.time_step > batch_size:
    value_coefficient = 0.1 / (Map.time_step - batch_size)
else:
    value_coefficient = 0.1 / (Map.time_step + 1)

loss =  entropy_coefficient * entropy_loss + value_coefficient * value_loss
```

** The entropy loss is calculated using the cross-entropy formula as we have our logits and target logits. (With this loss, the agent will favour the movement with more reward)

** The value loss is calculated by multiplying the (probability of performing the actions) and (the rewards gotten after performing that action). (With this loss, the agent will favour the current movement if the reward is positive)

- Lastly, this function will update the model using the loss and return the loss for observing each of the loss value:

```
loss.backward()          # used to compute & store the gradient in each neuron
optimizer.step()         # Performs a single optimization step
```

```
return loss, entropy_coefficient * entropy_loss, value_coefficient * value_loss
```

Now, we can start training the model as we have all the function that required to train the model. First, we will need to set the parameters and filename:

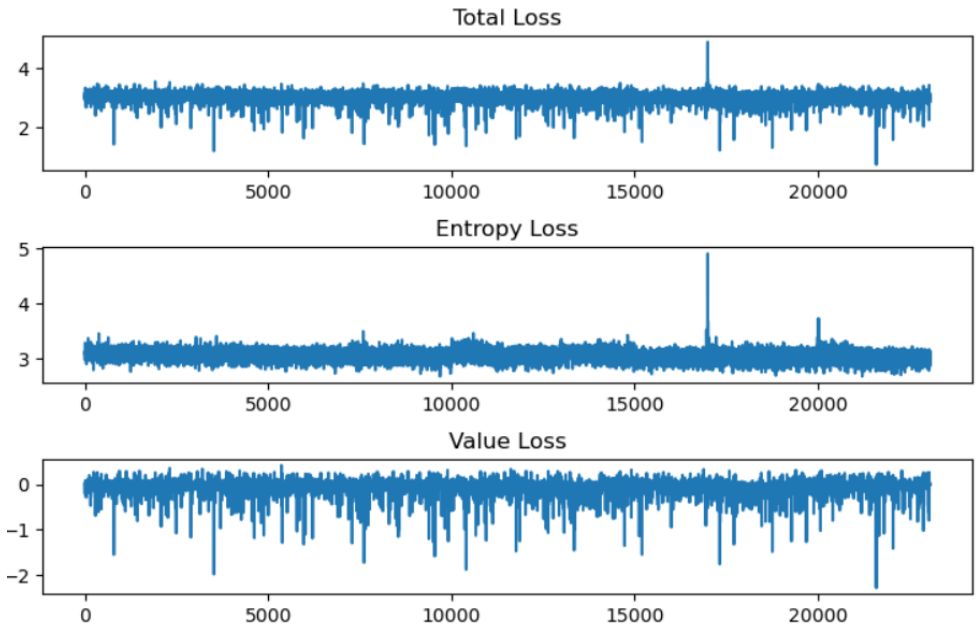
```
# Parameter and file name
BATCH_SIZE = 16
lr         = 0.00001          # learning rate
num_round  = 2                # number of round for each map
num_map    = 100              # number of map used
RDC        = 3                # Reward decrease count
PATH       = "CNN40000.pt"    # Path for the model
load_model = True              # Load model or not
MAX_STEP_ALLOW     = 100000   # Maximum step allowed for a map
MIN_REWARD_ALLOW   = -95      # Minimum reward allowed for a mop
num_epoch_per_map = 2000      # Total number of epoch per map
```

Then, we can train our model using the for loop:

```
for round_num in range(num_round):
    print(f"> > > > > > ROUND {round_num} < < < < <")
    for map_num in range(num_map):
        MAP_PATH = f"Maps\map{map_num}_trans.txt"
        Map = Environment(MAP_PATH, False)
        print(f"> > > FOR MAP {map_num} < < <")

        for i in range(num_epoch_per_map // num_round):

            loss, en_loss, va_loss = train(Map, optimizer, BATCH_SIZE, device, MAX_STEP_ALLOW, MIN_REWARD_ALLOW, RDC = RDC)

            if i % 100 == 0:
                print(f"After epoch {i} current position =", Map.curr_R, Map.curr_C,"Target =", Map.target, Map.time_step)
                print(f"   with loss {loss.cpu().detach().item():.5f} (entropy_loss) {en_loss.cpu().detach().item():.5f}",
                      f"(value_loss) {va_loss.cpu().detach().item():.5f}")


            batch_loss.append((loss.detach().tolist(),  en_loss.detach().tolist(), va_loss.detach().tolist()))

        if (map_num +1) % 10 == 0:  # Save the weight of the model for every 10 maps
            current_step = str(int(int(start_step) + num_epoch_per_map / (num_round - round_num) * (map_num+1))  )
            new_PATH = PATH.replace(start_step, current_step)
            torch.save(model.state_dict(), new_PATH)
            print("The model is save successfully with name", new_PATH)
```

** Note that for every 10 maps, we will save a checkpoint, so that we can stop training if we need to use the computer for doing other things.

Here is the example of the loss function gotten for training the model:



E. Effort on trying to overfit a map

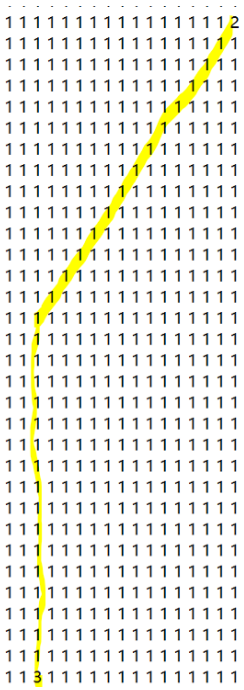
Since we have the value loss which can favor the current movement, ideally, we can overfit a map by fixing the action:

In [10]:
```
"""   FOR FIXING ACTION   """
# action_list = np.array([0, 0, 0, 0, 0,  7, 7, 7, 7, 7, 7, 7 ,  0, 0] )
# print(action_list)
def train(Map, optimizer, batch_size, device, max_step = 2000, min_reward = -95.0, RDC = 1):
```

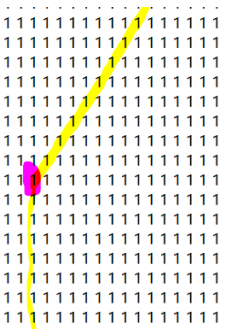
Then, replace it with the action gotten from get_action_n_prob() function

```
    # collect experience by acting in the environment with current policy
    while True:

        # Get an action and act in the environment
        OH_RC_obs = OH_RC_obs.to(device)
        va_move   = va_move.to(device)
        target_p, p, action = get_action_n_prob(OH_RC_obs, va_move, Map, model)  # action is an integer in range [0, 7]
        if device == torch.device("cuda"):
            OH_RC_obs = OH_RC_obs.to("cpu")
            va_move   = va_move.to("cpu")

#         action = action_list[Map.time_step]                          # Try fix action
#         print(action, target_p, Map.curr_R,Map.curr_C, Map.target)
```

However, after many try, we can't overfit a map. The issue is as follows. Consider a map where 2 is starting point and 3 is the target and the best path will be sequence of action [5] * 14 + [6] *16



The problem is at the turning point:



At the turning point, the observation is same as before. Therefore, it is impossible for a model to produce difference output given that the same input. Of course, we can consider using the time step to feed into the model. However, we do not explore further on it as it is very time consuming.

F. Evaluate the model on untrained map

- For evaluating, we have created 2 functions to help in evaluate the map

No	Function	Description
1	actions_to_target(R, C, target_R, target_C, Map)	return sequence of action that will reach the target if the agent observe the target
2	run_map(Model_path, Map_path, MAX_STEP_ALLOW = 150000)	Given a model path and map_path, output the batch of actions and save into the text file

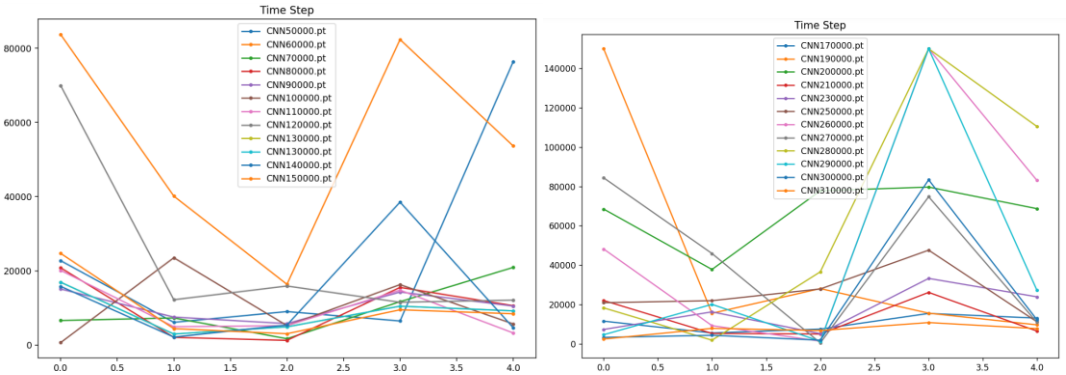
** For actions_to_target() function, the idea is that it will go through a while loop that keep looping and select the actions that can move nearer to the target until it reach target

** For run_map() function, it is similar to the train process, but in this case, we only get the observations, valid move, and actions (does not get logits of action)

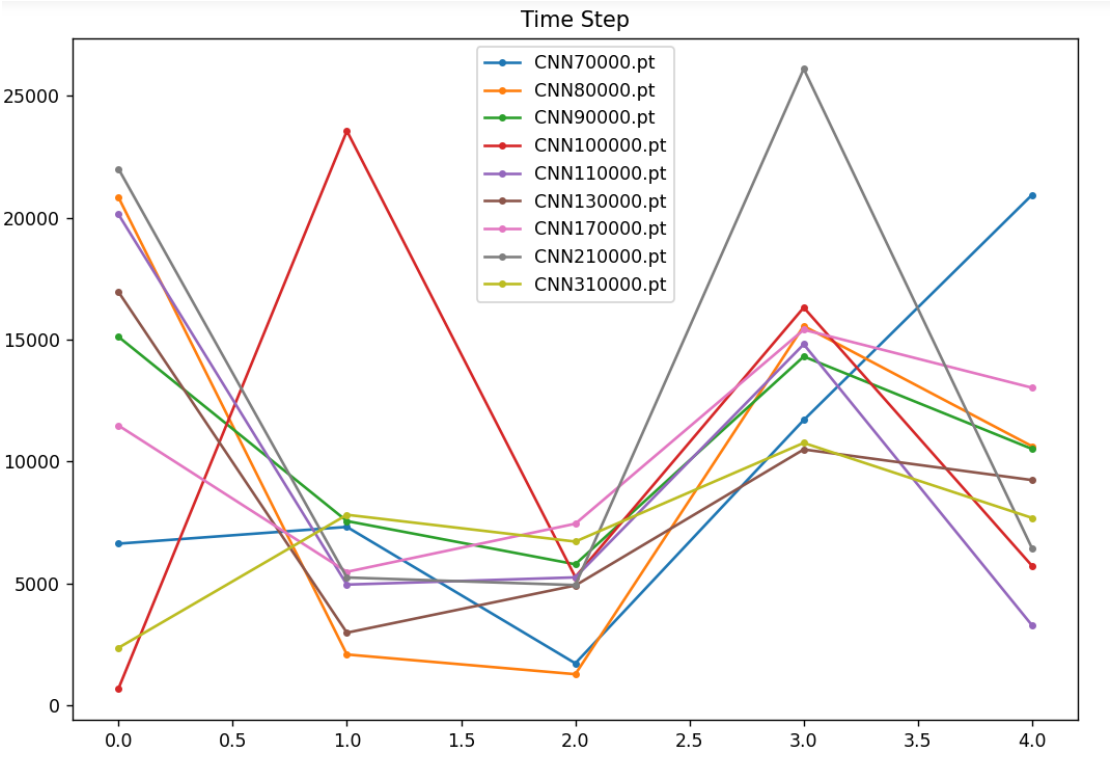
- The reason of writing function run_map() is to make us easy evaluating many map with difference model with only 1 block of command by changing the Model_path and Map_path

```
Model_path =  ["CNN40000.pt", "CNN50000.pt", "CNN60000.pt"]
Map_path = [f"Maps/map{i}_trans.txt" for i in range(40,60)]
# Map_path = ["Maps/map40_trans.txt", "Maps/map41_trans.txt", "Maps/map42_trans.txt", "Maps/map43_trans.txt", "Maps/map44_trans.txt"]

batch_cost_n_step = []
for map_p in Map_path:
    for model_p in Model_path:
        print(f"> > > > > For model '{model_p}' at map '{map_p}' < < < < <")
        cost, step = run_map(model_p, map_p)
        batch_cost_n_step.append([cost, step])
```

- We have evaluated 26 model (CNN50000 to CN31000) for 4 maps, and the result is shown at below:



Among them, the best model will be these models (the model at figure below). However, these models are not guaranteed to be the best as for different map, different model will have different performance

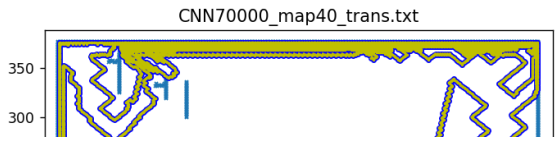


➢ After that, we can plot the map out using the obs_plot() in the Environment class.

Plot out the actions of the model in map

In [8]:
```
%matplotlib notebook
Model_path = ["CNN70000.pt", "CNN80000.pt", "CNN90000.pt", "CNN100000.pt", "CNN110000.pt","CNN130000.pt","CNN170000.pt","CNN210000.pt", "C
Map_path = [f"Maps/map{i}_trans.txt" for i in range(40,45)]

for map_p in Map_path:
    for model_p in Model_path:
        try:   # If there exist '/'
            idx = map_p.index('/')
            map_name = map_p[idx + 1:]
        except:  # If not
            map_name = map_p

        txt_path = f"Record_Action/{model_p[:-3]}_{map_name[:-4]}.txt"
        loaded_batch_action = np.genfromtxt(txt_path,delimiter=',',filling_values=0)[:-1]
        Map = Environment(map_p,False)
        Map.obs_plot(loaded_batch_action, txt_path[txt_path.index('/')+1:], False)
```

<IPython.core.display.Javascript object>



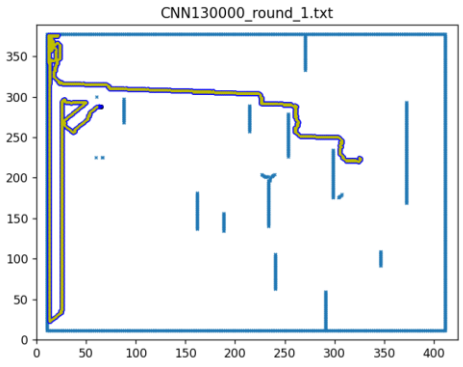
G. Use the model to participate the competition

- In the competition, we have use model
  - "CNN130000.pt" for round 1

- "CNN60000.pt" for round 2
- "CNN50000.pt" for round 3

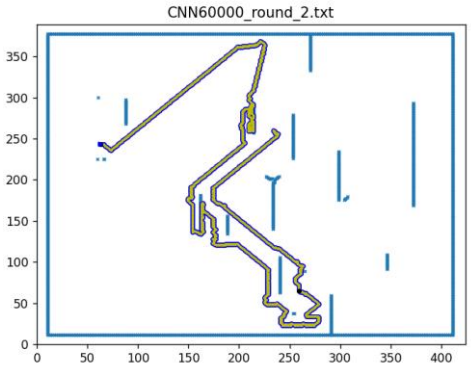
i) For round 1:

Time	Path

Round 1
In [14]: %%time
Model_path = ["CNN130000.pt"]
Map_path = ["round_1.txt"]
batch_cost_n_step = []
for map_p in Map_path:
for model_p in Model_path:
print(f"> > > > > For model '{model_p}' at map '{map_p}' < < < < <")
cost, step = run_map(model_p, map_p)
batch_cost_n_step.append([cost, step])
> > > > > For model 'CNN130000.pt' at map 'round_1.txt' < < < < <
The Target is At:
[[65 288]]
num_iteration to find correct path = 10
We find target at 65 288 ! We are at 55 282
Therefore, we will perform the following sequence of action [7, 7, 7, 7, 7, 7, 6, 6, 6, 6]
> > > The Game end! with cost 16804 (1324 step)
For this map, the number that the agent pass through a same position = [0 1 2 1000]
CPU times: total: 18.8 s
Wall time: 3.17 s
	

ii) For Round 2

Time	Path

Round 2
In [16]: %%time
Model_path = ["CNN60000.pt"]
Map_path = ["round_2.txt"]
batch_cost_n_step = []
for map_p in Map_path:
for model_p in Model_path:
print(f"> > > > > For model '{model_p}' at map '{map_p}' < < < < <")
cost, step = run_map(model_p, map_p)
batch_cost_n_step.append([cost, step])
> > > > > For model 'CNN60000.pt' at map 'round_2.txt' < < < < <
The Target is At:
[[63 243]
[239 65]]
num_iteration to find correct path = 10
We find target at 239 65 ! We are at 256 76
Therefore, we will perform the following sequence of action [5, 5, 5, 4, 4, 4, 4, 4, 4, 4]
num_iteration to find correct path = 10
We find target at 63 243 ! We are at 73 236
Therefore, we will perform the following sequence of action [1, 1, 1, 1, 1, 1, 1, 2, 2, 2]
> > > The Game end! with cost 13760 (1044 step)
For this map, the number that the agent pass through a same position = [0 1 1000]
CPU times: total: 9.09 s
Wall time: 1.66 s
	

iii) For round 3

Time	Path