In [1]:

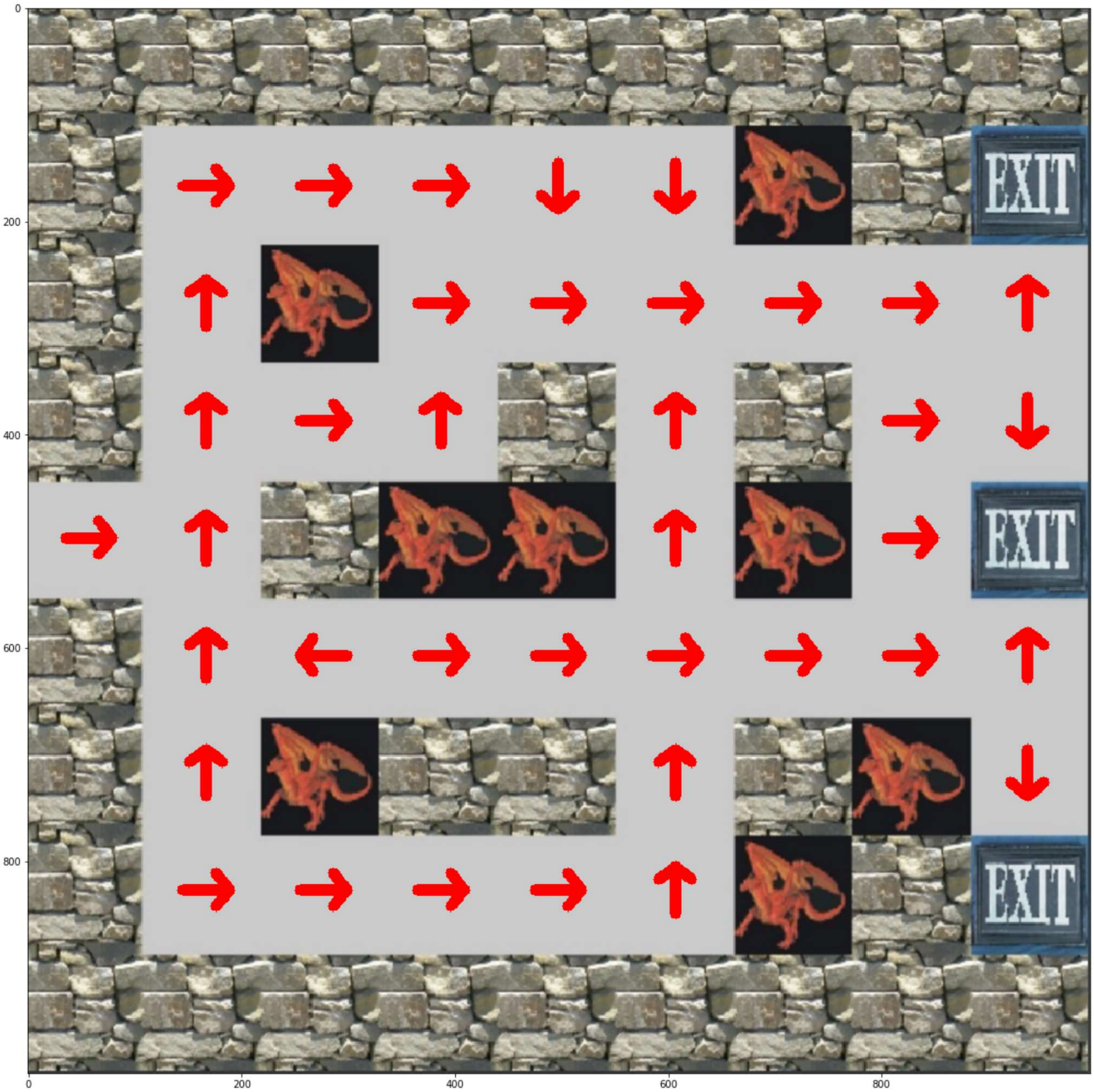
```python
import numpy as np
from collections import defaultdict

# Step 1: Read files
'''
Rewards = 81-d column array
Probs = dict{action : {s:[(s1', p1),...]}}
'''

REWARD_FILE = 'rewards.txt'
P_FILE = 'prob_a.txt'

def read_rewards(fname):
    with open(REWARD_FILE) as f:
        return np.mat([int(line.strip()) for line in f]).T

def read_probs(fname):
    probs = dict()
    for action in range(1, 5):
        probs[action] = defaultdict(list)
        name, ext = fname.split('.')
        filename = name + str(action) + '.' + ext
        with open(filename) as f:
            for line in f:
                content = line.split()
                s_cur, s_next, p = int(content[0]), int(content[1]), float(content[2
])
                probs[action][s_cur].append((s_next, p))
    return probs

rewards = read_rewards(REWARD_FILE)
probs = read_probs(P_FILE)

# Step 2: Initialize variables
gamma = 0.99
states = range(1, 82)
actions = range(1, 5)
policy = {state: np.random.randint(low = 1, high = 5) for state in states}

# Step 3: Start policy iteration

def evaluate_values():
    M = np.eye(len(states))
    for state in states:
        action = policy[state]
        for s_next, p in probs[action][state]:
            M[state - 1, s_next - 1] -= gamma * p
    return np.linalg.solve(M, rewards)

# values = np.mat() : MINUS 1
# Evaluate Values
def q_sa(state, action, values):
    reward = 0
    for s_next, p in probs[action][state]:
        reward += p * values[s_next - 1]
    return rewards[state - 1] + gamma * reward

# Greedy update policy
def update_policy(values):
    is_updated = False
    policy_new = {state: None for state in states}
    for state in states:
        q_max, action_best = float('-inf'), None
```

```python
        for action in actions:
            q_sa_value = q_sa(state, action, values)
            if q_max < q_sa_value:
                q_max, action_best = q_sa_value, action
        policy_new[state] = action_best
        if action_best != policy[state]:
            is_updated = True
    return is_updated, policy_new

is_updated = True
iter = 0
while is_updated:
    values = evaluate_values()
    is_updated, policy = update_policy(values)
    iter += 1
print("Iteration = {}".format(iter))

best_value = evaluate_values().reshape((9, 9)).T
print(best_value)
# np.savetxt('bestvalue.txt', best_value, fmt='%g')
best_policy = np.array([action for _, action in sorted(list(policy.items()))]).reshap
e((9, 9)).T
print(best_policy)
# np.savetxt('bestpolicy.txt', best_policy, fmt='%g')
```

```
Iteration = 4
[[   0.            0.            0.            0.            0.
     0.            0.            0.            0.        ]
 [   0.           65.77308407   67.13647421   77.84605      79.84451583
    72.47511769 -100.            0.          100.        ]
 [   0.           55.88294346 -100.           70.30818136   81.34440225
    83.04847989   84.88054612   96.87232244   98.71875987]
 [   0.           54.92298013   50.47656297   59.66641187    0.
    80.95826449    0.           97.04482865   98.72729893]
 [  53.50968756   54.14557214    0.         -100.         -100.
    61.77980767 -100.           88.22035599  100.        ]
 [   0.           52.50402036   43.9359876    51.09137525   61.00715483
    71.78642614   73.94661407   85.18458536   97.57257319]
 [   0.           43.77254574 -100.            0.            0.
    70.35142939    0.         -100.           88.40593622]
 [   0.           47.95296148   48.76871928   58.14735126   59.39003194
    60.1688947  -100.            0.          100.        ]
 [   0.            0.            0.            0.            0.
     0.            0.            0.            0.        ]]
[[1 1 1 1 1 1 1 1 1]
 [1 3 3 3 4 4 1 1 1]
 [1 2 1 3 3 3 3 3 2]
 [1 2 3 2 1 2 1 3 4]
 [3 2 1 1 1 2 1 3 1]
 [1 2 1 3 3 3 3 3 2]
 [1 2 1 1 1 2 1 1 4]
 [1 3 3 3 3 2 1 1 1]
 [1 1 1 1 1 1 1 1 1]]
```

In [2]:

```python
%matplotlib inline
import math
import numpy as np

import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
maze = mpimg.imread('Maze.jpg')
plt.rcParams['figure.figsize'] = [20, 20]

def draw_arrow(start, end, color, thickness, image, length = 20, alpha = 127):
    PI = 3.1415926
    angle = math.atan2(end[1] - start[1], end[0] - start[0])
    cv2.line(image, start, end, color, thickness)

    arrow_x = end[0] + length * math.cos(angle + PI * alpha / 180)
    arrow_y = end[1] + length * math.sin(angle + PI * alpha / 180)
    cv2.line(image, (int(arrow_x), int(arrow_y)), end, color, thickness)

    arrow_x = end[0] + length * math.cos(angle - PI * alpha / 180)
    arrow_y = end[1] + length * math.sin(angle - PI * alpha / 180)
    cv2.line(image, (int(arrow_x), int(arrow_y)), end, color, thickness)

def draw_direction(x, y, direction, image, color, thickness):
    width = 22
    side = 110
    x_center, y_center = x * side + width, y * side + width
    if direction == 1:
        start, end = (x_center + width, y_center), (x_center - width, y_center)
    elif direction == 2:
        start, end = (x_center, y_center + width), (x_center, y_center - width)
    elif direction == 3:
        start, end = (x_center - width, y_center), (x_center + width, y_center)
    elif direction == 4:
        start, end = (x_center, y_center - width), (x_center, y_center + width)


    biasx = biasy = 35
    start = (start[0] + biasx, start[1] + biasx)
    end = (end[0] + biasy, end[1] + biasy)
    draw_arrow(start, end, color, thickness, image)

numbered_square = set([5, 20, 22, 24 ,26, 29, 30, 31, 33, 35, 38, 39, 42, 44, 57, 60,
 75, 76, 78, 79] + list(range(66, 70)) + list(range(47, 54)) + list(range(11, 18)))
def is_numbered_square(x, y):
    place = x * 9 + y + 1
    return place in numbered_square

color = (255,0,0)
thickness = 10
arrow_image = np.copy(maze)
for x in range(9):
    for y in range(9):
        if is_numbered_square(x, y):
            draw_direction(x, y, best_policy[y, x], arrow_image, color, thickness)
plt.imshow(arrow_image)
plt.show()
```


0
200
400
600
800
EXIT
EXIT
EXIT
0
200
400
600
800

In [3]:
```
def value_iteration():
    v_t = np.array([0.0] * len(states))
    delta = float('inf')
    threshhold = 0.0001
    while delta > threshhold:
        delta = 0
        for state in states:
            best_value = float('-inf')
            for action in actions:
                best_value = max(best_value, rewards[state - 1] + gamma * sum(v_t[s_n
ext - 1] * p for s_next, p in probs[action][state]))
            delta = max(delta, abs(best_value - v_t[state - 1]))
            v_t[state - 1] = best_value
#         print(v_t)
    return v_t


v_t_star = value_iteration()
v_star = evaluate_values().T
print(v_t_star)
print(v_star)
print("State value function in part a and b are {}".format("the same" if np.all((abs(
v_t_star - v_star)<1e-2)) else "different"))
```

```
[  0.           0.           0.           0.          53.50369907
   0.           0.           0.           0.           0.
  65.76590103  55.87677976  54.91691772  54.13958365  52.49819804
  43.76763184  47.94774456   0.           0.          67.12923121
 -99.99015743  50.47110082   0.          43.93106347 -99.99015743
  48.76348165   0.           0.          77.83779421  70.30077743
  59.66008697 -99.99015743  51.08596413   0.          58.14122806
   0.           0.          79.8361461   81.3359714    0.
 -99.99015743  61.00082068   0.          59.38385357   0.
   0.          72.46759065  83.03997576  80.94995886  61.77343094
  71.77908872  70.34422529  60.16270949   0.           0.
 -99.99015743  84.87195914   0.         -99.99015743  73.93914581
   0.         -99.99015743   0.           0.           0.
  96.86264728  97.03507372  88.21156951  85.17609259 -99.99015743
   0.           0.           0.          99.99015743  98.70902034
  98.71747665  99.99015743  97.56294932  88.3971411   99.99015743
   0.        ]
[[   0.            0.            0.            0.           53.50968756
     0.            0.            0.            0.            0.
    65.77308407   55.88294346   54.92298013   54.14557214   52.50402036
    43.77254574   47.95296148    0.            0.           67.13647421
  -100.           50.47656297    0.           43.9359876  -100.
    48.76871928    0.            0.           77.84605      70.30818136
    59.66641187 -100.           51.09137525    0.           58.14735126
     0.            0.           79.84451583   81.34440225    0.
  -100.           61.00715483    0.           59.39003194    0.
     0.           72.47511769   83.04847989   80.95826449   61.77980767
    71.78642614   70.35142939   60.1688947     0.            0.
  -100.           84.88054612    0.         -100.           73.94661407
     0.         -100.            0.            0.            0.
    96.87232244   97.04482865   88.22035599   85.18458536 -100.
     0.            0.            0.          100.           98.71875987
    98.72729893  100.           97.57257319   88.40593622  100.
     0.        ]]
State value function in part a and b are the same
```