Question 1:
Note: in my implementation, 3*3 grid is represented by an array of 9. Cross player represented by -1, circle player represented by 1, empty space represented by 0

Trajectory 1:
State:
```
[[ 0.  0.  0.]
 [ 0.  0. -1.]
 [ 0.  0.  0.]]
```
Reward to go: 10.0

State:
```
[[ 1.  0.  0.]
 [ 0. -1. -1.]
 [ 0.  0.  0.]]
```
Reward to go: 10.0

State:
```
[[ 1.  0.  0.]
 [ 0. -1. -1.]
 [ 1. -1.  0.]]
```
Reward to go: 10.0

State:
```
[[ 1.  0. -1.]
 [ 0. -1. -1.]
 [ 1. -1.  1.]]
```
Reward to go: 10.0

State:
```
[[ 1.  0. -1.]
 [ 1. -1. -1.]
 [ 1. -1.  1.]]
```
Reward to go: 10.0

—————————————

Trajectory 2:
State:
```
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0. -1.]]
```
Reward to go: 3.439

State:
```
[[ 0.  0.  0.]
 [ 0.  0. -1.]
 [ 0.  1. -1.]]
```
Reward to go: 2.71

State:
```
[[ 1.  0.  0.]
 [ 0. -1. -1.]
 [ 0.  1. -1.]]
```

Reward to go: 1.9

State:
[[1. -1. 0.]
 [1. -1. -1.]
 [0. 1. -1.]]
Reward to go: 1.0

——————————————-

Trajectory 3
State:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. -1.]]
Reward to go: -4.580000000000001

State:
[[0. 1. 0.]
 [0. -1. 0.]
 [0. 0. -1.]]
Reward to go: -6.200000000000001

State:
[[1. 1. -1.]
 [0. -1. 0.]
 [0. 0. -1.]]
Reward to go: -8.0

State:
[[1. 1. -1.]
 [1. -1. 0.]
 [-1. 0. -1.]]
Reward to go: -10.0

————————————————-

Trajectory 4
State:
[[1. 0. 0.]
 [0. -1. 0.]
 [-1. 0. 0.]]
Reward to go: 2.71

State:
[[1. 0. 0.]
 [0. -1. 0.]
 [-1. 0. -1.]]
Reward to go: 1.9

State:
[[1. 0. 1.]
 [0. -1. 0.]

[-1. 0. -1.]]
Reward to go: 1.0

———————————————-

Trajectory 5

State:
[[0. 0. 0.]
 [-1. 0. 0.]
 [0. 0. 0.]]
Reward to go: -3.122

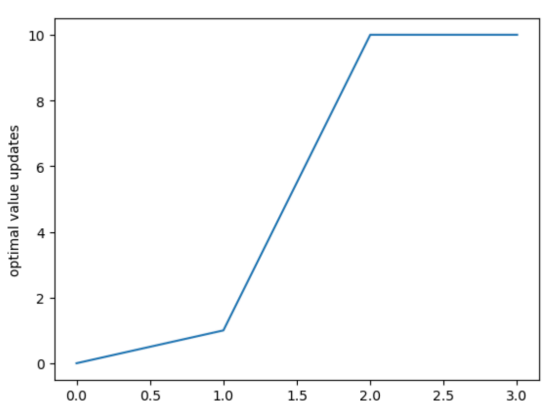
State:
[[0. 0. -1.]
 [-1. 0. 1.]
 [0. 0. 0.]]
Reward to go: -4.580000000000001

State:
[[0. 0. -1.]
 [-1. -1. 1.]
 [0. 0. 1.]]
Reward to go: -6.200000000000001

State:
[[1. 0. -1.]
 [-1. -1. 1.]
 [0. -1. 1.]]
Reward to go: -8.0

State:
[[1. -1. -1.]
 [-1. -1. 1.]
 [1. -1. 1.]]
Reward to go: -10.0

Q2



Q3:

The values are no more than 0.1 different from the optimal values of any state because value iteration defines a contraction mapping $||B(V) - B(V')||\infty \leq \gamma||V - V'||\infty$, where γ is the contraction rate. A contraction mapping has a unique fixed point that it converges to, so value iteration converges to a unique solution in the value space, as seen in this implementation of value iteration, where $||V - V'||\infty$ approaches and eventually reaches zero after around 6 iterations.

Q4

```
S1_1 = np.array([[0, 0, 0], [0, 0, -1], [0, 0, 0]])
optimal_values, actions = value_interation(S1_1,0.1,np.array2string(S1_1))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 8.6878
action pos: (0, 0)
```

```
S1_2 = np.array([[1, 0, 0], [0, -1, -1], [0, 0, 0]])
optimal_values, actions = value_interation(S1_2,0.1,np.array2string(S1_2))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 8.541999999999998
action pos: (1, 0)
```

```
S1_3 = np.array([[1, 0, 0], [0, -1, -1], [1, -1, 0]])
optimal_values, actions = value_interation(S1_3,0.1,np.array2string(S1_3))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 10.0
action pos: (1, 0)
```

```
S1_4 = np.array([[1, 0, -1], [0, -1, -1], [1, -1, 1]])
optimal_values, actions = value_interation(S1_3,0.1,np.array2string(S1_3))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 10.0
action pos: (1, 0)
```

```
S1_5 = np.array([[1, 0, -1], [1, -1, -1], [1, -1, 1]])
optimal_values, actions = value_interation(S1_5,0.1,np.array2string(S1_5))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 10.0
action pos: (0, 1)
```

```
S2_1 = np.array([[0, 0, 0], [0, 0, 0], [0, 0, -1]])
optimal_values, actions = value_interation(S2_1,0.1,np.array2string(S2_1))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 8.6878
action pos: (1, 1)
```

```
S2_2 = np.array([[0, 0, 0], [0, 0, -1], [0, 1, -1]])
optimal_values, actions = value_interation(S2_2,0.1,np.array2string(S2_2))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 6.562
action pos: (1, 1)
```

```
S2_3 = np.array([[1, 0, 0], [0, -1, -1], [0, 1, -1]])
optimal_values, actions = value_interation(S2_3,0.1,np.array2string(S2_3))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 4.27
action pos: (0, 2)
```

```
S2_4 = np.array([[1, -1, 0], [1, -1, -1], [0, 1, -1]])
optimal_values, actions = value_interation(S2_4,0.1,np.array2string(S2_4))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 10.0
action pos: (2, 0)
```

```
S3_1 = np.array([[0, 0, 0], [0, 0, 0], [0, 0, -1]])
optimal_values, actions = value_interation(S3_1,0.1,np.array2string(S3_1))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 8.6878
action pos: (1, 1)
```

```
S3_2 = np.array([[0, 1, 0], [0, -1, 0], [0, 0, -1]])
optimal_values, actions = value_interation(S3_2,0.1,np.array2string(S3_2))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 7.948
action pos: (0, 0)
```

```python
S3_3 = np.array([[1, 1, -1], [0, -1, 0], [0, 0, -1]])
optimal_values, actions = value_interation(S3_3,0.1,np.array2string(S3_3))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 4.27
action pos: (2, 0)
```

```python
S3_4 = np.array([[1, 1, -1], [1, -1, 0], [-1, 0, -1]])
optimal_values, actions = value_interation(S3_4,0.1,np.array2string(S3_4))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: -10.0
action pos: None
```

```python
S4_1 = np.array([[1, 0, 0], [0, -1, 0], [-1, 0, 0]])
optimal_values, actions = value_interation(S4_1,0.1,np.array2string(S4_1))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 8.541999999999998
action pos: (0, 2)
```

```python
S4_2 = np.array([[1, 0, 0], [0, -1, 0], [-1, 1, -1]])
optimal_values, actions = value_interation(S4_2,0.1,np.array2string(S4_2))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 7.569999999999999
action pos: (0, 2)
```

```python
S4_3 = np.array([[1, 0, 1], [0, -1, -1], [-1, 1, -1]])
optimal_values, actions = value_interation(S4_3,0.1,np.array2string(S4_3))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 10.0
action pos: (0, 1)
```

```python
S5_1 = np.array([[0, 0, 0], [-1, 0, 0], [0, 0, 0]])
optimal_values, actions = value_interation(S5_1,0.1,np.array2string(S5_1))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 8.687800000000003
action pos: (0, 2)
```

```
S5_2 = np.array([[0, 0, -1], [-1, 0, 1], [0, 0, 0]])
optimal_values, actions = value_interation(S5_2,0.1,np.array2string(S5_2))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 7.219
action pos: (1, 1)
```

```
S5_3 = np.array([[0, 0, -1], [-1, -1, 1], [0, 0, 1]])
optimal_values, actions = value_interation(S5_3,0.1,np.array2string(S5_3))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: 7.569999999999999
action pos: (2, 0)
```

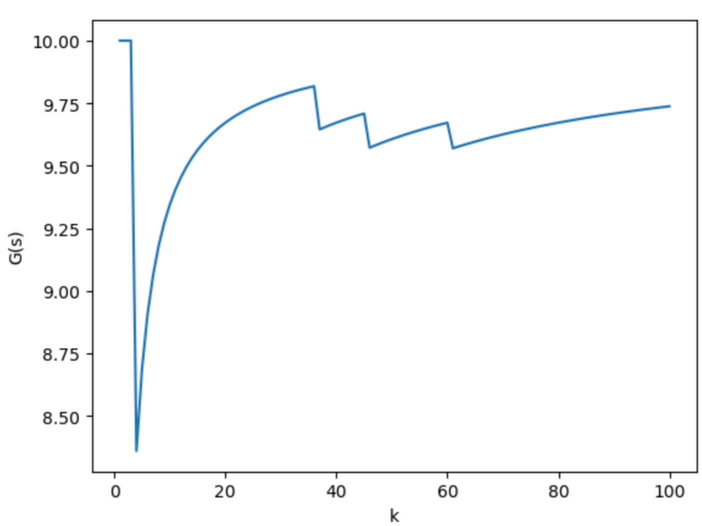
```
S5_4 = np.array([[1, 0, -1], [-1, -1, 1], [0, -1, 1]])
optimal_values, actions = value_interation(S5_4,0.1,np.array2string(S5_4))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: -8.0
action pos: (0, 1)
```

```
S5_5 = np.array([[1, -1, -1], [-1, -1, 1], [1, -1, 1]])
optimal_values, actions = value_interation(S5_5,0.1,np.array2string(S5_5))
print("optimal_v:",optimal_values[-1])
print("action pos:",actions)
```

```
optimal_v: -10
action pos: None
```

Q5
State = [[-1, 0, 0], [0, 0, 0], [0, 0, 0]])



Q6
States:
Every car space(14 cells) in the middle is a state

Action:
1.Move one cell down
2.Move two cell down
3.Move diagonal down(down left or down right)

Reward:
10 in P1 cell or P2 cell
-1 for any other cells in the two lane
-100 for drive outside two lanes

Transition:
1, since it is a deterministic environment, an action will result in a definite state

γ: 0.9

Other parameter for Q learning:

ϵ = 0.2

α = 0.8

Q7

A1:Move one cell down: 6.2
A2:Move two cell down: 6.2
A3:Move diagonal down(down left or down right):6.2

Q8

States:
Every car cell spaces and the position of the two pedestrians combination is a state, that is 14*(8+1)*(8+1) states(+1 for out of sight)

Action:
1.Move one cell down
2.Move two cell down
3.Move diagonal down(down left or down right)

Reward:
10 for car in P1 cell or P2 cell
-1 for any other car cells
-1000 for hitting pedestrian

Transition:
1, since it is a deterministic environment, an action will result in a definite state

γ: 0.9

Other parameter for Q learning:

ϵ = 0.2

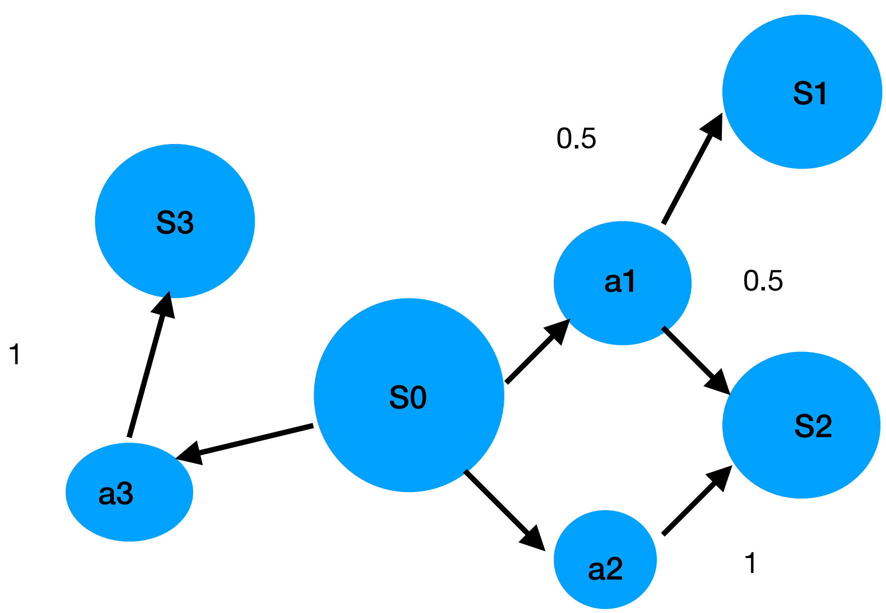
α = 0.8

Q9

A1:Move one cell down: -901
A2:Move two cell down: -901
A3:Move diagonal down(down left or down right):4.58

Q12

1.



2. Q-value function: $Q(s0, a) = 3s0 + a$

Reward function: $R(s0) = α$, $R(s1) = 0$, $R(s2) = 4β$, $R(s3) = 6β$

$Q(s0, a1) = α + β$

$Q(s0, a1) = R(s0) + γ(0.5R(s1) + 0.5R(s2)) = R(s0) + 0.25R(s1) + 0.25R(s2) = α+0.25(0)+0.25(4β)= α+ β$

$Q(s0,a2) = α + 2β$

$Q(s0,a2) = R(s0) + γR(s2) = R(s0) + 0.5R(s2) = α + 0.5(4β) = α + 2β$

$Q(s0,a3) = α + 3β$
$Q(s0,a3) = R(s0) + γR(s3) = R(s0) + 0.5R(s3) = α + 0.5(6β) = α + 3β$

3. There are no α, β values in the linear Q approximation function that can represent the Q-values at $s0$ since these Q-values do not represent a linear relationship.
Reward function: $R(s0) = α$, $R(s1) =- 4β$, $R(s2) = 8β$, $R(s3) = 18β$
$Q(s0,a1) = R(s0) + 0.25R(s1) + 0.25R(s2) = α + 0.25(- 4β) + 0.25(8β) = α + β$ $Q(s0,a2) = R(s0) + 0.5R(s2) = α + 0.5(8β) = α + 4β$
$Q(s0,a3) = R(s0) + 0.5R(s3) = α + 0.5(18β) = α + 9β$

4. Q-value function: $Q(s0, a) = α + a2β$
Reward function: $R(s0) = α$, $R(s1) =- 4β$, $R(s2) = 8β$, $R(s3) = 18β$ $Q(s0,a1)= α+ β= α+ β$
$Q(s0,a2)= α+22β= α+4β$
$Q(s0,a3)= α+32β= α+9β$
This approximation function matches the Q-values from above.