1 Requirements

In [7]:

```
#CODE
```

2 Exercises

2.1 Solve Exercise 3.4. Explain how you obtained the table. Your solution may be hand-written.

Let R_{search} be the rv of reward when the robot is searching.
$P_{R_{search}}(r) = P[R_{search} = r]$

Let R_{wait} be the rv of reward when the robot is waiting.
$P_{R_{wait}}(r) = P[R_{wait} = r]$

Let there distribution be s.t.
$E[R_{search}] = r_{search}$
$E[R_{wait}] = r_{wait}$

Then,

s	a	s'	r	p(s',r|s,a)
high	search	high	r	$\alpha \, . \, P_{R_{search}}(r)$
high	search	low	r	$(1-\alpha) \, . \, P_{R_{search}}(r)$
high	wait	high	r	$P_{R_{wait}}(r)$
low	search	high	-3	$(1-\beta)$
low	search	low	r	$\beta \, . \, P_{R_{search}}(r)$
low	wait	low	r	$P_{R_{wait}}(r)$
low	recharge	high	0	1.0

2.2 Write code that solves the linear equations required to find $v_\pi(s)$ and generate the values in the table in Figure 3.2. Note that the policy π picks all valid actions in a state with equal probability. Add comments to your code that explain all your steps.

In [1]:

```
#CODE
```

2.3 Solve Exercises 3.15 and 3.16.

$v_\pi(s) = E_\pi[G_t | S_t = s]$

2.3.1 Solve Exercises 3.15

Continuous

$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots | S_t = s]$

Adding c to all rewards

$$v^c_\pi(s) = E_\pi[(R_{t+1} + c) + \gamma(R_{t+2} + c) + \gamma^2(R_{t+3} + c) \dots | S_t = s]$$
$$= E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots | S_t = s] + E_\pi[c + \gamma c + \gamma^2 c \dots | S_t = s]$$
$$= v_\pi(s) + \frac{c}{1-\gamma}$$

The value function doesn't change relatively for states. Each of them have same scalar added to them.

2.3.2 Solve Exercises 3.16

Episodic

$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{n-1} R_{t+n} | S_t = s]$

Adding c to all rewards

$$v^c_\pi(s) = E_\pi[(R_{t+1} + c) + \gamma(R_{t+2} + c) + \gamma^2(R_{t+3} + c) + \dots + \gamma^{n-1}(R_{t+n} + c) | S_t = s]$$
$$= E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{n-1} R_{t+n} | S_t = s] + E_\pi[c + \gamma c + \gamma^2 c + \dots + \gamma^{n-1} c | S_t = s]$$
$$= v_\pi(s) + c\frac{1-\gamma^n}{1-\gamma}$$

The value function does change relatively for states. Now it depends upon the time(n) after which the episode ends when started in state s. It increases as n increases. And as the time after which the episode ends for different start states, the value of the additive term changes and hence the value functions of the term aren't relatively same as before.

2.4 Write code that generates the optimal state-value function and the optimal policy for the Gridworld in Figure 3.5. You want to solve the corresponding system of non-linear equations. Explain all your steps.

In [2]:

```
#CODE
```

2.5 Given an equation for v∗ in terms of q∗.

$$q_*(s,a) = \max_{\pi} q_\pi(s,a)$$
$$v_*(s) = \max_{a \in A(s)} \max_{\pi} q_\pi(s,a)$$
$$v_*(s) = \max_{a \in A(s)} q_*(s,a)$$

2.6 Code policy iteration and value iteration (VI) to solve the Gridworld in Example 4.1. Your code must log output of each iteration. Pick up a few sample iterations to show policy evaluation and improvement at work. Similarly, show using a few obtained iterations that every iteration of VI improves the value function. Your code must include the fix to the bug mentioned in Exercise 4.4.

In [3]:

```
#CODE
```

2.6.1 Policy Iteration

The bug mentioned in Exercise 4.4 is dealt by using numpy.argamx() which internally handles it, as it has follows a consistent convention when selecting between equal values. It always selects the one that has a lower index and since indexes of actions don't change in code, therefor it can't oscillate between equally favaourable policies as it will always choose the action with a lower index.

In [4]:

```
#CODE
```

2.6.2 Value Iteration

In [5]:

```
#CODE
```

2.7 Code exercise 4.7.

In [6]:

```
#CODE
```