In [1]:
```python
from IPython.core.display import Image
Image("https://dl.dropbox.com/u/62929183/algorithm_mcl.png")
```

Out[1]:

Algorithm MCL($\mathcal{X}_{t-1}, u_t, z_t, m$):

$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$

for $m = 1$ to M do

$\quad x_t^{[m]} = \mathrm{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$

$\quad w_t^{[m]} = \mathrm{measurement\_model}(z_t, x_t^{[m]}, m)$

$\quad \bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$

endfor

for $m = 1$ to M do

$\quad$ draw i with probability $\propto w_t^{[i]}$

$\quad$ add $x_t^{[i]}$ to $\mathcal{X}_t$

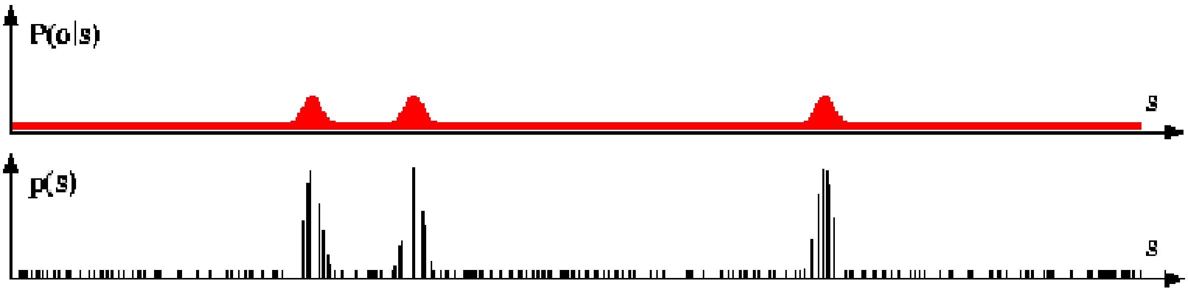
endfor

return $\mathcal{X}_t$

In [ ]:
```python
#- Ut -> Odometria (comando movimento)
#- Zt -> Sensores (observacao)
def monte_carlo_filter(Ut, Zt):
    for m in range(num_particulas):
        _Xt[m] = sample_motion_model(Ut, _Xt[m])
        _Wt[m] = measurement_model(Zt, _Xt[m], m)

    _Xt = resample()
    _Wt = [base_weight]*num_particulas
```

In [2]:
```python
from IPython.core.display import Image
Image("https://dl.dropbox.com/u/62929183/algorithm_mcl_resample.png")
```

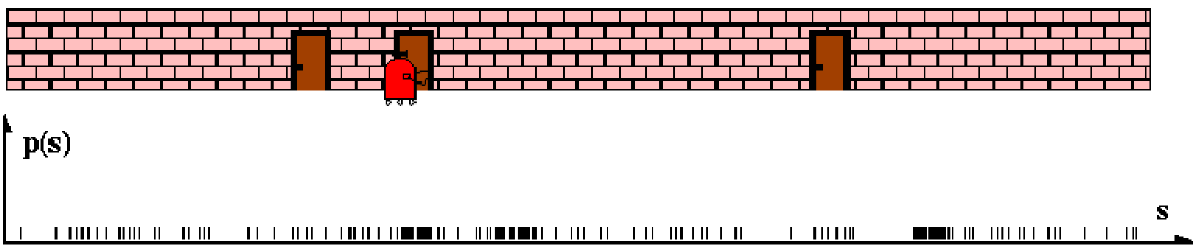
Out[2]:



In [ ]:
```python
def resample():
    '''
    usa os vetores _Xt e _Wt como sendo a p(z|x) X(barra)t
    '''
    #normaliza para fazer o sorteio
    normalize()

    #novas particulas
    Xt=[(0)]*num_particulas

    for m in range(num_particulas):
        i = draw_with_probability()
        Xt[m]=_Xt[i]

    return Xt
```

In [3]:
```python
from IPython.core.display import Image
Image("https://dl.dropbox.com/u/62929183/algorithm_mcl_resample_done.png")
```

Out[3]:



In [ ]:
```
#calcula nova posicao dado o comando de deslocamento
def sample_motion_model(Ut, Xt):
    #atua
    motion_pos = Xt+Ut
    #adiciona ruido
    motion_pos=noise(motion_pos)

    #valida particula
    #if(motion_pos>=max_ambiente or motion_pos<min_ambiente):
    #    #saiu do ambiente, gera nova particula aleatoria
    #    motion_pos = new_particle()

    #cilcula no ambiente
    if(motion_pos>max_ambiente):
        motion_pos = motion_pos - max_ambiente
    if(motion_pos<min_ambiente):
        motion_pos = motion_pos + max_ambiente

    return motion_pos
```

In [ ]:
```
#medicao
def measurement_model(Zt, Xt, m):
    ndx = int(Xt)
    seen = mundo[ndx]
    norm = Xt-ndx
    hit = 0.5
    if(seen==Zt):
        #TODO: correto seria usar a curva gaussiana
        if(norm>=0 and norm<0.1):
            hit=1.20
        if(norm>=0.1 and norm<0.2):
            hit=1.33
        if(norm>=0.2 and norm<0.3):
            hit=1.45
        if(norm>=0.3 and norm<0.4):
            hit=1.70
        if(norm>=0.4 and norm<0.6):
            hit=2.00
        if(norm>=0.6 and norm<0.7):
            hit=1.70
        if(norm>=0.7 and norm<0.8):
            hit=1.45
        if(norm>=0.8 and norm<0.9):
            hit=1.33
        if(norm>=0.9 and norm<=1):
            hit=1.20
    return base_weight*hit
```

In [ ]:
```
#- retorna um indice baseado no sorteio dentro da
#distribuicao de probabilidade (roleta)
def draw_with_probability():
    roleta = random.random()
    total = 0.0
    ndx = 0
    for i in range(num_particulas):
        total+=_Wt[i]
        if(total>=roleta):
            ndx = i
            break

    return ndx
```