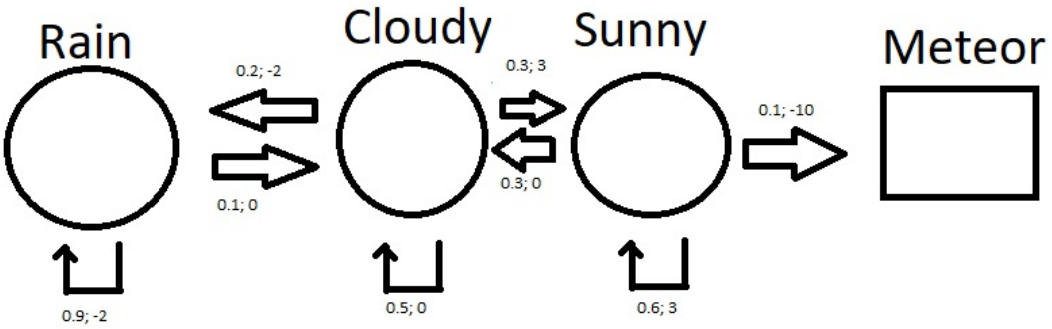
1.1 en 1.2



1.3

Rain Cloudy Sunny Meteor = 0*1 + 3* 1 -10*1 = -7

Rain Cloudy Sunny Cloudy Sunny Sunny Meteor = 0*1 + 3* 1 + 1*0 + 1*3 + 1*3 – 10*1 = -1

1.4

De formules zijn terug te lezen in 1_4.py

State	V0	V1	V2
Rain	0	-1.8	-3.37
Cloudy	0	0.5	0.316
Sunny	0	0.8	1.375
Meteor	0	0	0

1.5

Een discount van 1 kan zorgen voor een oneindige loop. Een reward die oneindig, praktisch oneindig of gewoon “erg ver” in de toekomst zit telt net zo zwaar mee als een reward nu. Deze reward moet dus ook worden meegenomen en zo kan er geen reward bepaald worden omdat deze reward zover weg zit dat hij (praktisch niet berekend kan worden.

Ook zal een algoritme zo lang mogelijk door blijven gaan als acties een positieve reward hebben. Stel we maken een taxi ai en elke keer dat deze ai niet tegen een boom aanrijd krijgt het 5 punten als het op de locatie aankomt krijgt het 500 punten. Deze ai zal oneindig lang blijven rijden en niet tegen bomen aanrijden en nooit naar de finish gaan. Die 500 punten is namelijk een gegeven die krijgt het toch of het gelijk ernaar toe gaat of niet. Als de discount value lager is dan 1 dan zijn die 500 punten geen gegeven meer. Hoe langer de ai wacht hoe lager de reward. De ai zal dus willen eindigen

Een ander punt is dat in financiële situaties een reward nu kan worden geprefereerd over een reward later omdat er rente kan worden verdient over de reward nu. Uit onderzoek blijk dat dit ook in zijn algemeenheid geld. Mensen en dieren tonen een bereidheid om een lagere reward nu te nemen in plaats van een hogere reward later.

De reward in de toekomst is vaak onzeker omdat er van alles kan veranderen waardoor die reward niet exact is zoals wij denken dat hij is.

Als mensen prefereren wij dus reward nu tegenover een reward later. Om dit gedrag na te bootsen in een markov procces gebruiken wij een discount die lager is dan 1 zodat het algoritme ook in bepaalden situaties een reward nu prefereert over een reward later.

2

Laten we de states s1, s2 en s3 noemen. Laten we beginnen bij s3.

s3 kan geen acties uitvoeren dus blijft de value 0.

s2 kan naar links voor -1 (-1 + 0) of naar rechts voor -0.1(-0.1 + 0). s2 krijgt de value -0.1.

s1 kan naar rechts en dit levert -0.1 + 0 = -0.1 op s1 wordt -0.1.

s2 kan naar links voor -1 (-1 + 0) of naar rechts voor -0.2(-0.1 + -0.1). s2 krijgt de value -0.2.

Dit gaat even door zie hier onder

Iteratie	S1	S2	S3
0	0	0	-1
1	-0.1	-0.1	-1
2	-0.2	-0.2	-1
3	-0.3	-0.3	-1
4	-0.4	-0.4	-1
5	-0.5	-0.5	-1
6	-0.6	-0.6	-1
7	-0.7	-0.7	-1
8	-0.8	-0.8	-1
9	-0.9	-0.9	-1
10	-1	-1	-1
11	-1.1	-1	-1
12	-1.1	-1	-1

Na iteratie 12 zullen er geen veranderingen meer zijn.

Omdat de value van s3 niet meer veranderd en s2 altijd naar s3 zal gaan zal de value van s2 niet meer veranderen.

Omdat s1 altijd naar s2 gaat zal s1 ook niet meer veranderen.