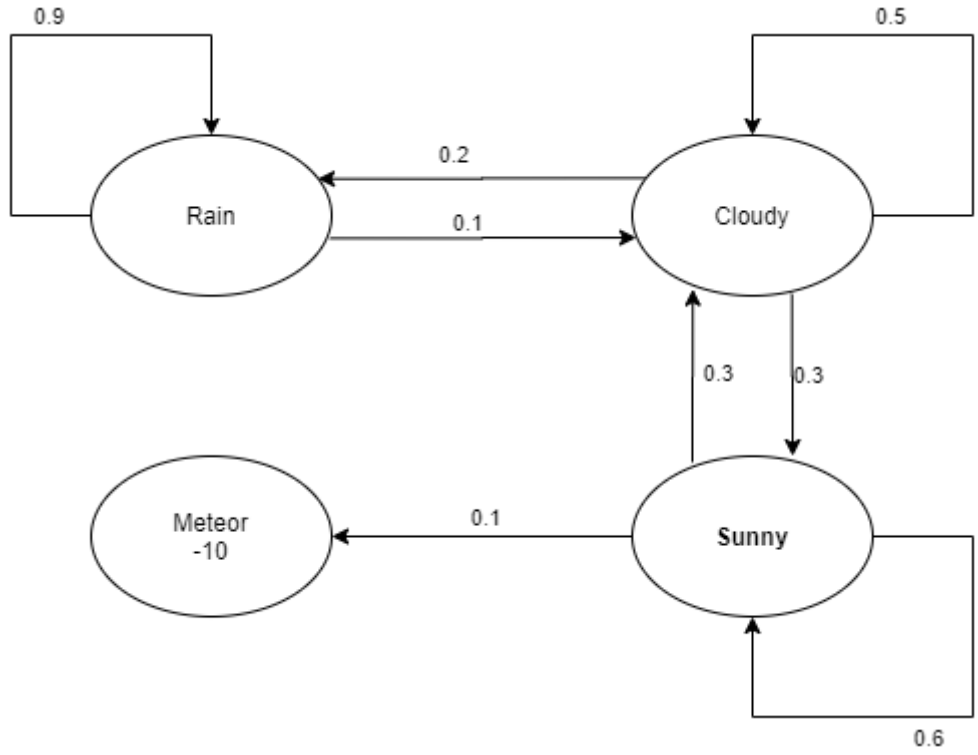
Opdracht 1.1
Adaptive Systems

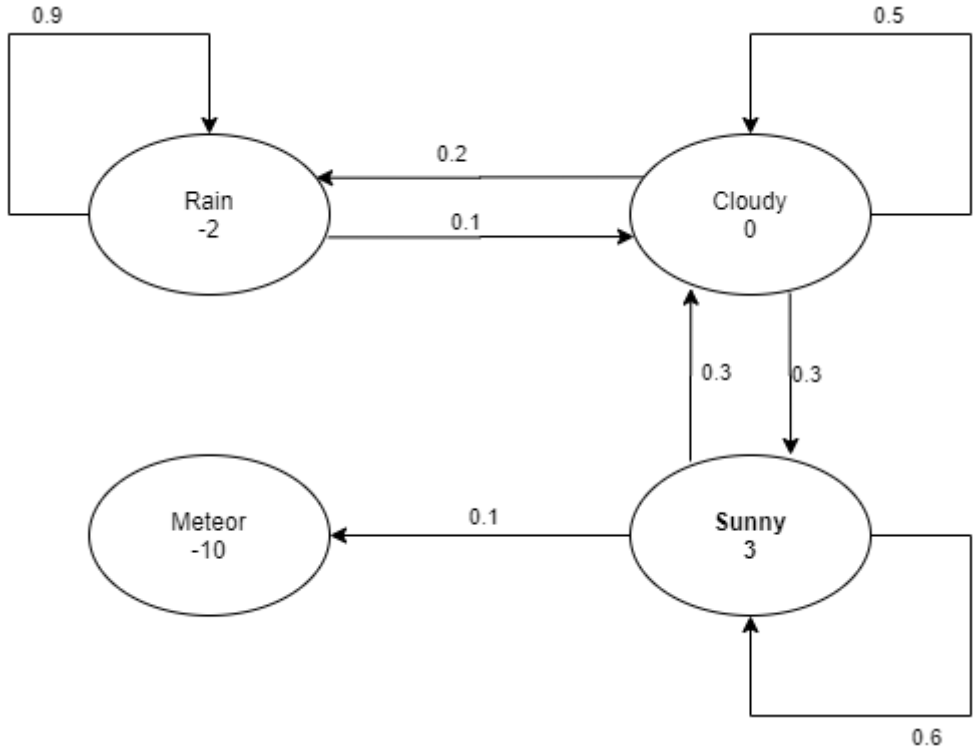
Student:	Storm Joannes
Studentnummer:	1760581
Opleiding:	HBO-ICT Artificial Intelligence
Instelling:	Hogeschool Utrecht
Code:	2022_TICT_VINNO1-33_3_V
Datum:	22-01-2024

A. Markov Chain



...

B. Markov Reward Proces



...

C. Sampling

De samples zijn berekend met een discount van 1.

Sample 1: Cloudy → Rainy → Cloudy → Sunny → Meteor
G_5: 0 - 2 * 1 + 0 * 1 + 3 * 1 - 10 * 1 = -9

Sample 2: Rainy → Rainy → Cloudy → Sunny → Sunny → Meteor
G_6: -2 - 2 * 1 + 0 * 1 + 3 * 1 + 3 * 1 - 10 * 1 = -8

D. Value function

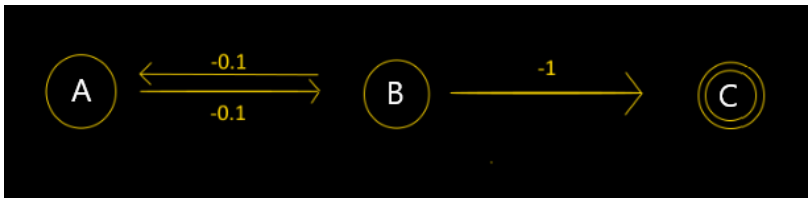
Iteratie 1:
V_r: -2 + 1 * ((0.1 * 0) + (0.9 * 0)) = -2
V_c: 0 + 1 * ((0.2 * 0) + (0.5 * 0) + (0.3 * 0)) = 0
V_s: 3 + 1 * ((0.6 * 0) + (0.3 * 0) + (0.1 * 0)) = 3
V_m: -10

Iteratie 2:
V_r: -2 + 1 * ((0.1 * 0) + (0.9 * -2)) = -3.8
V_c: 0 + 1 * ((0.2 * -2) + (0.5 * 0) + (0.3 * 3)) = 0.5
V_s: 3 + 1 * ((0.6 * 3) + (0.3 * 0) + (0.1 * -10)) = 3.8
V_m: -10

Iteration	Rain	Cloud	Sunny	Meteor
0	0	0	0	0
1	-2	0	3	-10
2	-3.8	0.5	3.8	-10

Discount:
1. Als γ gelijk is aan 1 worden toekomstige beloningen niet verminderd. Dit kan zorgen voor een oneindige beloning.
2. Bij een discount van 1 kijkt hij te veel naar het belang van toekomstige beloningen. Dit kan resulteren in langzame convergentie of instabiliteit.

E. Value iteration



Discount = 1
Beginwaarde van alle states = 0

Iteration 1:
A = -0.1 + 1 * 0 = -0.1
B = Max((-0.1 + 1 * 0 = -0.1), (-1 + 1 * 0 = -1)) = -0.1

Iteration 2:
A = -0.1 + 1 * -0.1 = -0.2
B = Max((-0.1 + 1 * -0.1), (-1 + 1 * 0)) = -0.2

Iteration 3:
A = -0.1 + 1 * -0.2 = -0.3
B = Max((-0.1 + 1 * -0.2), (-1 + 1 * 0)) = -0.3

Iteration 4:
A = -0.1 + 1 * -0.3 = -0.4
B = Max((-0.1 + 1 * -0.3), (-1 + 1 * 0)) = -0.4

Iteration 5:
A = -0.1 + 1 * -0.4 = -0.5
B = Max((-0.1 + 1 * -0.4), (-1 + 1 * 0)) = -0.5

Iteration 6:
A = -0.1 + 1 * -0.5 = -0.6
B = Max((-0.1 + 1 * -0.5), (-1 + 1 * 0)) = -0.6

Iteration 7:
A = -0.1 + 1 * -0.6 = -0.7
B = Max((-0.1 + 1 * -0.6), (-1 + 1 * 0)) = -0.7

Iteration 8:
A = -0.1 + 1 * -0.7 = -0.8
B = Max((-0.1 + 1 * -0.7), (-1 + 1 * 0)) = -0.8

Iteration 9:
A = -0.1 + 1 * -0.8 = -0.9
B = Max((-0.1 + 1 * -0.8), (-1 + 1 * 0)) = -0.9

Iteration 10:
A = -0.1 + 1 * -0.9 = -1
B = Max((-0.1 + 1 * -0.9), (-1 + 1 * 0)) = -1

Iteration 11:
A = -0.1 + 1 * -1 = -1.1
B = Max((-0.1 + 1 * -1), (-1 + 1 * 0)) = -1

Iteration	V(s): A → B	V(s): B → A	V(s): B → C	Value A	Value B	Value C
0	0	0	-1	0	0	0
1	-0.1	-0.1	-1	-0.1	-0.1	0
2	-0.2	-0.2	-1	-0.2	-0.2	0
3	-0.3	-0.3	-1	-0.3	-0.3	0
4	-0.4	-0.4	-1	-0.4	-0.4	0
5	-0.5	-0.5	-1	-0.5	-0.5	0
6	-0.6	-0.6	-1	-0.6	-0.6	0
7	-0.7	-0.7	-1	-0.7	-0.7	0
8	-0.8	-0.8	-1	-0.8	-0.8	0
9	-0.9	-0.9	-1	-0.9	-0.9	0
10	-1	-1	-1	-1	-1	0
11	-1.1	-1.1	-1	-1.1	-1	0

Conclusie
In de tabel hierboven is te zien dat er een aantal stappen mogelijk zijn. Van A naar B, van B naar A, en van B naar C. Alleen bij B is er dus een keuze beschikbaar voor wat de beste volgende state is. Door de greedy policy toe te passen zal B altijd kiezen voor de stap waarbij de waarde zo hoog mogelijk is.

Je kunt zien dat state A tot de tiende iteratie de hogere waarde heeft, waardoor state B hier dan ook voor zal kiezen tijdens die iteratie. Echter is te zien dat vanaf iteratie tien de waarde om van state B naar state A steeds verder in de min gaat, terwijl de waarde van state B naar state C op een constante -1 staat.

Dit betekent dat bij iteratie 10 state B een gelijke keuze heeft om naar state A of state C te gaan (licht blauw gearceerd), maar dat het itereren stopt na iteratie 11 aangezien hier de stap van B naar C de betere stap is, en C een eind state is.