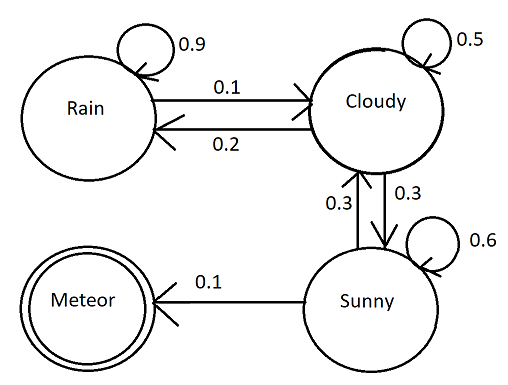
AS inleveropdracht 1

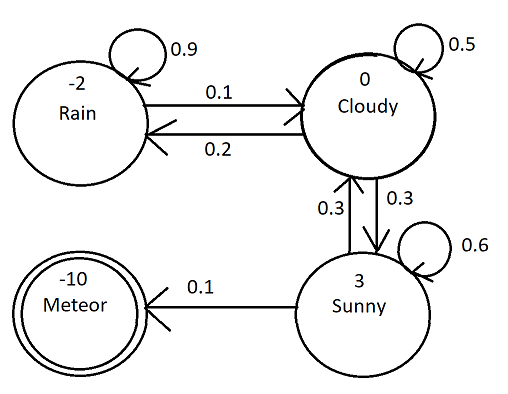
Thijme de Bruijn

# Opdracht 1: Prediction

## 1.1: Markov Chain



## 1.2: Markov Process Reward



## 1.3: Sampling. Een voorbereiding voor Monte-Carlo Policy Evaluation

Afbeelding met tekst, horloge

Automatisch gegenereerde beschrijving

Laten we de twee samples maken:

1. Rain -> Cloudy -> Sunny -> Meteor
2. Sunny -> Sunny -> Cloudy -> Sunny -> Sunny - > Meteor

Eerst berekenen we de return Gt van sample 1:

Gt = -2 + 0 + 3 + -10 = -9

Nu berekenen we de return Gt van sample 2:

Gt = 3 + 3 + 0 + 3 + 3 + -10 = 2

## 1.4: De value-function bepalen

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## 1.5: Zelf onderzoek

De vraag hier is wat het probleem kan zijn wanneer we de discount factor op 1 houden.

Hier zijn twee problemen mee:

Probleem 1:

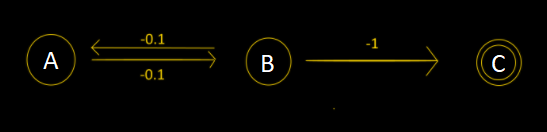
Het doel van de discount factor is om ervoor te zorgen dat berekende staten in de toekomst minder zwaar meetellen. De onzekerheid dat je deze staat bereikt, wordt dus meeberekent. Dus bij één stap in de toekomst heb je bijvoorbeeld een discount factor van 0.9. Bij twee stappen in de toekomst heb je 0.9 \* 0.9 = 0.81. Bij drie stappen 0.9 \* 0.9 \* 0.9 = 0.729 enzovoorts.

Het probleem wat je hebt wanneer je een discount factor hebt van 1 is dat het niet uitmaakt hoe ver je in de toekomst bent, de discount factor blijft 1.

Probleem 2:

Bij bijvoorbeeld policy iteration wordt de discount factor gebruikt om de value van een staat mee te vermenigvuldigen. Alleen zal dit altijd hetzelfde zijn als de value van die staat. 1 \* A = A. Dus in andere woorden. Het bestaan van de discount function in de formule als deze 1 is, heeft helemaal geen invloed en is dus nutteloos.

# Opdracht 2: Control met Value Iteration

  
Figuur 1: State model

In figuur 1 zie je het model waar wij mee beginnen. In dit model zijn alle values nul. Ik ga nu ‘Value iteration’ gebruiken om te kijken welke values de staten zouden moeten hebben. In deze hele iteratie zal Y = 1 blijven.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iteration | V(s): A -> B | V(s): B -> A | V(s): B -> C | Max policy A | Max policy B | Value A | Value B | Value C |
| 0 |  |  |  | A -> B | B -> A | 0 | 0 | 0 |
| 1 | -0.1 + 1 \* -0 = -0.1 | -0.1 + 1 \* -0 = -0.1 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.1 | -0.1 | 0 |
| 2 | -0.1 + 1 \* -0.1 = -0.2 | -0.1 + 1 \* -0.1 = -0.2 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.2 | -0.2 | 0 |
| 3 | -0.1 + 1 \* -0.2 = -0.3 | -0.1 + 1 \* -0.2 = -0.3 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.3 | -0.3 | 0 |
| 4 | -0.1 + 1 \* -0.3 = -0.4 | -0.1 + 1 \* -0.3 = -0.4 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.4 | -0.4 | 0 |
| 5 | -0.1 + 1 \* -0.4 = -0.5 | -0.1 + 1 \* -0.4 = -0.5 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.5 | -0.5 | 0 |
| 6 | -0.1 + 1 \* -0.5 = -0.6 | -0.1 + 1 \* -0.5 = -0.6 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.6 | -0.6 | 0 |
| 7 | -0.1 + 1 \* -0.6 = -0.7 | -0.1 + 1 \* -0.6 = -0.7 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.7 | -0.7 | 0 |
| 8 | -0.1 + 1 \* -0.7 = -0.8 | -0.1 + 1 \* -0.7 = -0.8 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.8 | -0.8 | 0 |
| 9 | -0.1 + 1 \* -0.8 = -0.9 | -0.1 + 1 \* -0.8 = -0.9 | -1 + 1 \* 0 = -1 | A -> B | B -> A | -0.9 | -0.9 | 0 |
| 10 | -0.1 + 1 \* -0.9 = -1 | -0.1 + 1 \* -0.9 = -1 | -1 + 1 \* 0 = -1 | A -> B | B -> A, B -> C | -1 | -1 | 0 |
| 11 | -0.1 + 1 \* -1 = -1.1 | -0.1 + 1 \* -1 = -1.1 | -1 + 1 \* 0 = -1 | A -> B | B -> C | -1.1 | -1 | 0 |
| 12 | -0.1 + 1 \* -1 = -1.1 | -0.1 + 1 \* -1.1 = -1.2 | -1 + 1 \* 0 = -1 | A -> B | B -> C | -1.1 | -1 | 0 |

Tabel 1: Value Iteration proces

In Tabel 1 zie je hoe onze value iteration is verlopen. Zoals je kan zien, is het voor state A alleen maar mogelijk om naar B te gaan. Dus state A zal dit altijd blijven doen. State B heeft daartegenover twee opties. Hij kan naar A of naar C gaan. Zoals je in iteratie 1 ziet, is het voor B meer profitabel om naar A te gaan. Vanaf dit moment zal de kost van B naar A elke iteratie omlaaggaan met 0.1. Ditzelfde patroon blijft zich herhalen tot iteratie 10. In iteratie 10 is het net zo profitabel, voor state B, om naar state A als naar state C te gaan. Maar als we nog een iteratie doorgaan, zal het opeens meer profitabel zijn voor state B om naar state C te gaan dan naar state A. Ik ben nog één iteratie doorgegaan en ik heb hier gevonden dat de values van de staten niet meer veranderen. Dit is een teken dat we de werkelijke waardes van de staten hebben bereikt en het zal niet meer nodig zijn om verdere iteraties te berekenen. Want de policy’s zullen hetzelfde blijven.

# Opdracht 3: Implementatie

## 3.1: Het betoverde doolhof