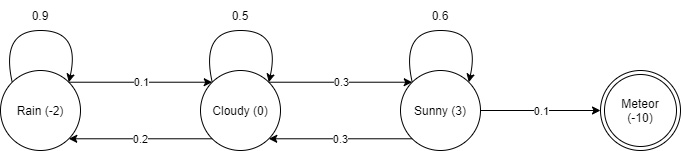
# Prediction

* 1. Markov Chain:
  2. Markov Reward Process:



* 1. Sampling. Een voorbereiding voor Monte-Carlo Policy Evaluation:

Rain->Cloudy->Sunny->Meteor = (-2) + (0) + (3) + (-10) = -9

Rain->Rain->Cloudy ->Sunny->Sunny->Cloudy->Sunny->Meteor = (-2) + (-2) + (0) + (3) + (3) +(0) + (3) + (-10) = -5

* 1. De value-function bepalen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Iteratie** | **Rain** | **Cloudy** | **Sunny** | **Meteor** |
| **0** | 0 | 0 | 0 | 0 |
| **1** | -1.8 | 0.5 | 0.8 | 0 |
| **2** | -3.37 | 0.63 | 1.43 | 0 |

Berekeningen:

|  |  |
| --- | --- |
|  | **Iteratie 1** |
| **Rain** | (0.9 \* (-2 + 1 \* 0)) + (0.1 \* (0 + 1 \* 0)) = -1.8 |
| **Cloudy** | (0.5 \* (0 + 1 \* 0)) + (0.2 \* (-2 + 1 \* 0)) + (0.3 \* (3 + 1 \* 0)) = 0.5 |
| **Sunny** | (0.6 \* (3 + 1 \* 0)) + (0.3 \* (0 + 1 \* 0)) + (0.1 \* (-10 + 1 \* 0)) = 0.8 |
| **Meteor** | N.V.T. |

|  |  |
| --- | --- |
|  | **Iteratie 2** |
| **Rain** | (0.9 \* (-2 + 1 \* -1.8)) + (0.1 \* (0 + 1 \* 0.5)) = -3.37 |
| **Cloudy** | (0.5 \* (0 + 1 \* 0.5)) + (0.2 \* (-2 + 1 \* -1.8)) + (0.3 \* (3 + 1 \* 0.8)) = 0.63 |
| **Sunny** | (0.6 \* (3 + 1 \* 0.8)) + (0.3 \* (0 + 1 \* 0.5)) + (0.1 \* (-10 + 1 \* 0)) = 1.43 |
| **Meteor** | N.V.T. |

* 1. Zelf-onderzoek:

Als je een discount factor van 1 hebt zeg je dat de toekomst altijd even zwaar meeweegt als je huidige situatie, dit zou niet logisch zijn want bij je huidige situatie heb je al zekerheid en door te zeggen dat het even zwaar meetelt impliceer je dat je huidige beslissing niet uitmaakt want je kijkt even veel naar de toekomst.

Door 1 als discount factor te hebben is een state 3 stappen in de toekomst 1 ^ 3 = 1 van belang, dit zou niet moeten kloppen want in het echt is er vaak na mate er meer tijd verloopt meer onzekerheid.

Bovendien als je 1 als discount factor hebt kan een policy (mogelijk) niet convergeren.

# Control met Value Iteration



|  |  |  |  |
| --- | --- | --- | --- |
| **Iteratie** | **A** | **B** | **C** |
| **0** | 0 | 0 | 0 |
| **1** | -0.1 | -0.5 | 0 |
| **2** | -0.6 | -0.5 | 0 |
| **3** | -0.6 | -0.5 | 0 |

We zien dat het met een k=2 al convergeert, deze waardes zijn ongunstig omdat je met een greedy policy niet bij de eindstand terecht kan komen. Bij positie A is er maar een keuze om naar B te gaan maar bij B is er de keuze om naar A of C te gaan. A heeft een waarde van -0.6-0.1=-0.7 en C heeft een waarde van -1 dus zal hij altijd naar B gaan en vervolgend dan heen en weer tussen A en B.

# Implementatie

* 1. Ik heb de state class vervangen met een MazeCell class omdat gezien de simulatie mij logischer leek.
  2. Zie agent.py.
  3. Geïmplementeerd, een MazeCell kan een slagingskans van een actie hebben zoals aangegeven. Ik heb voor het voorbeeld en de simpliciteit toegevoegd dat als je 0.7 meegeeft elke directie een 70% slagingskans heeft en dus 3 x 10% kans om een andere kant op te gaan dan verwacht. Dit zou makkelijk uitgebreid kunnen worden om elke richting een andere slagingskans te geven.