# Verslag labo 2

1. **Question 1: Value Iteration**

Placeholder.

1. **Question 2: Bridge Crossing Analysis**

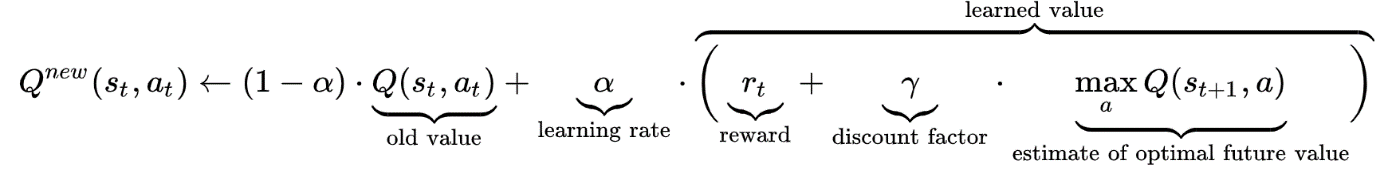
Placeholder.

1. **Question 3: Policies**

Placeholder.

1. **Question 6: Q-Learning**

Q-Learning, in tegenstelling tot Value Iteration in opdracht 1, leert wel uit ervaring. In een Q-tabel wordt, voor elke combinatie van action en state, een Q-value bijgehouden. In het begin zijn deze waarden 0. Na een aantal keer spelen, wordt de tabel stelselmatig aangevuld. De Agent kiest zijn volgende actie door het zoeken van de hoogste Q-value in de huidige state.

In “QLearningAgent” werden een aantal methodes vervolledigd. “Learning” in Q-Learning gebeurt voornamelijk in update(state, action, nextState, reward). Hier krijgt een action, state paar een nieuwe Q-value aangewezen. De Agent kan deze later gebruiken voor het maken van een beslissing. De gebruikt formule is gebaseerd op deze van Wikipedia:

De laatste term in de vorige formule werd geïmplementeerd in “computeValueFromQValues”. De Agent gaat altijd naar de volgende state met de hoogste Q-Value. Elke Q-value, na het nemen van een actie, wordt in een lijst gestoken. Na het doorlopen van alle “legale” acties, kan de maximum Q-value van de lijst teruggegeven worden aan het programma. “ComputeActionfromQValues” geeft de actie terug, bijhorend bij de maximum Q-value.

1. **Question 7: Epsilon Greedy**

In de Q-Learning Agent ontbreekt één functie: “getAction(state)”. Tijdens het programma loopt, zijn er twee opties mogelijk. In epsilon keer van de gevallen, geeft de functie een willekeurige actie terug (uit “legalActions”). Epsilon kunnen we veranderen naar wens in de command line. “util.flipcoin(p)” geeft True terug met kans p (hier epsilon). In alle andere gevallen geven we de beste actie terug met “ComputeActionfromQValues”. Deze implementatie hebben we reeds gezien in vorige opdracht.

1. **Question 8: Bridge Crossing Revisited**

Na het uitproberen van enkele combinaties (epsilon en learning-rate) werd het redelijk snel duidelijk dit na 50 iteraties niet mogelijk zou zijn. Het leren van een “optimal policy” met een kans van 99% is echter niet haalbaar. Met deze reden, geven we de string “NOT POSSIBLE” terug.

1. **Question 9: Q-Learning and Pacman**

Deze opdracht past Q-Learning toe in de wereld van Pacman. We leren hieruit dat Q-Learning effectief is in een redelijk kleine omgeving. In een groot doolhof, waarin Pacman meestal wordt gespeeld, duurt Q-Learning lang. Het geeft hierbij ook een inefficiënt resultaat.

1. **Question 10: Approximate Q-Learning**

Placeholder.