# Target

The whole project is to let the signal agent use RL to find the best signal strategy.

Signal Agent – Play the Traffic Game – Get Strategy table.

[State] – [Action] – [Reward Expectations]

PS: dumb car first, smart car later.

# ABM

The aim of the ABM is to make a **game environment** for signal agents to learn what is the best strategy when it’s facing different traffic environments.

## Environment Definition

Using the grid world ([Mesa SingleGrid](https://mesa.readthedocs.io/en/stable/apis/space.html#mesa.space.SingleGrid)), patches with color present the signal agent area.

**Grid attributes:**

State – road, signal, intersection, none - string

**Grid functions:**

isRoad – return Boolean

isSignal – return Boolean

isIntersection - Return Boolean

### Single Intersection Env

A picture containing text, square, screenshot, line

Description automatically generated

### Multiple Intersection Env

A screenshot of a computer game

Description automatically generated with low confidence

## Agent Definition

### Signal Agent

**Attribute:**

ID - int

State – store the signal current state – NS or EW – string.

Reward – the reward in each step - double

**Behavior:**

Switch – change the current state to opposite state, NS to EW or EW to NS

### Car Agent

**Attribute:**

Speed – how far a car can go in a step – double.

Tips – the reward which cars give to signal agent – double

Head – the cars heading – string

\* For model extension

Path – contain the direction in each intersection – list

Position – local position in the path – string

**Behavior:**

Move – move the car to his head based on its speed

Stop – set the speed in 0

SpeedUp – increase the speed

SpeedDown – decrease the speed

CostReward – Calculate the tips

GiveReward – give the reward to signal agent

Die – delete this car

### Generation Agent

This agent is used to generate cars from the out of intersection system.

Or add cars into the intersection system.

**Attribute:**

Code – The identification code – string? Or int

Location – its location relative to the intersection system – (X,Y)

GenSpeed – the speed that it creates car agent – int/ double

Behavior:

CreateCar – place a new car with attribute.

### Recycling Agent

Clean the car agent who left the system.

**Attribute:**

Code – The identification code – string? Or int

Location – its location relative to the intersection system – (X,Y)

**Behavior:**

DestoryCar – Return Boolean.

## Data Collector

Environment Data Collection:

Traffic flow in four directions – in tuple (N, S, W, E) - int

Average waiting time – double

Average speed – double

Current Reward – double

Expectation Reward Table – in tuple or matrix? – [State, Action, Reward]

# RL

Mainly using the Q Learning,

The task scenario is a typical Markovian decision process, so it is appropriate to use reinforcement learning to solve it, whose main consideration is the optimization idea of the Bellman Equation.

Markov Decision Process (MDP) is a mathematical model used to describe the environment, which assumes that the next state of the environment depends only on the current state and the actions taken, independent of the past states and actions. This is known as the Markov property, and the MDP usually consists of four elements: the set of states, the set of actions, the reward function, and the transfer probability function.

In the traffic signal control problem desired to be described, the state of the environment can be defined by the traffic flow and the traffic light state, the action is the switching of the signal, and the reward is the number of vehicles passing through the intersection at a given time. Given the current state and the action of the signal, the next state (i.e., the traffic flow and traffic light state at the next time step) depends only on the current state and action, and is independent of the past state and action. Thus, this problem can be modeled as a Markov decision process.

The Bellman Equation (BEE) is a formula used to describe the recursive relationship of the state value function or the state-action value function (also known as the Q function) in a Markovian decision process. For the state value function V(s), its Bellman equation is

For the state-action value function Q(s, a), its Bellman equation is

In these equations, s and s' denote the current state and the next state, respectively, a and a' denote the current action and the next action, respectively, r denotes the reward, π denotes the strategy, p denotes the state transfer probability, and γ is the discount factor.

The Bellman equation describes how the state value function or the state-action value function is related to the value function or the state-action value function of the next state given the policy and the state transfer probability. In reinforcement learning, we can gradually approximate the solution of the Bellman equation by iteratively updating the value function or state-action value function to find the optimal policy.

Traditional Q-Learning is suitable for discrete and limited scenarios, such as mazes. But for the scenario kind of traffic flow that we wish to use in this case is continuous, so we need to classify the traffic flow of the scenario into classes. The traffic flow in each direction is discretized into several classes, and then these classes are used as part of the state, and eventually the discrete and finite traffic flow in the four directions is incorporated into the decision about the traffic light.

Assuming that the traffic flow in each direction is discretized into three classes (low, medium and high), the states can be represented as a quintet: (north traffic class, south traffic class, east traffic class, west traffic class, traffic light state). In this case, the size of the Q table would be 3\*3\*3\*3\*2\*2 (because there are three classes of four traffic directions, two traffic light states and two actions), for a total of 324 state-action pairs.

Test Code: [Link](https://colab.research.google.com/drive/14q2Y8AQ5Y2NV2bHMmq8BWatBa0QanU7L#scrollTo=QI9XGb3EH8DY)