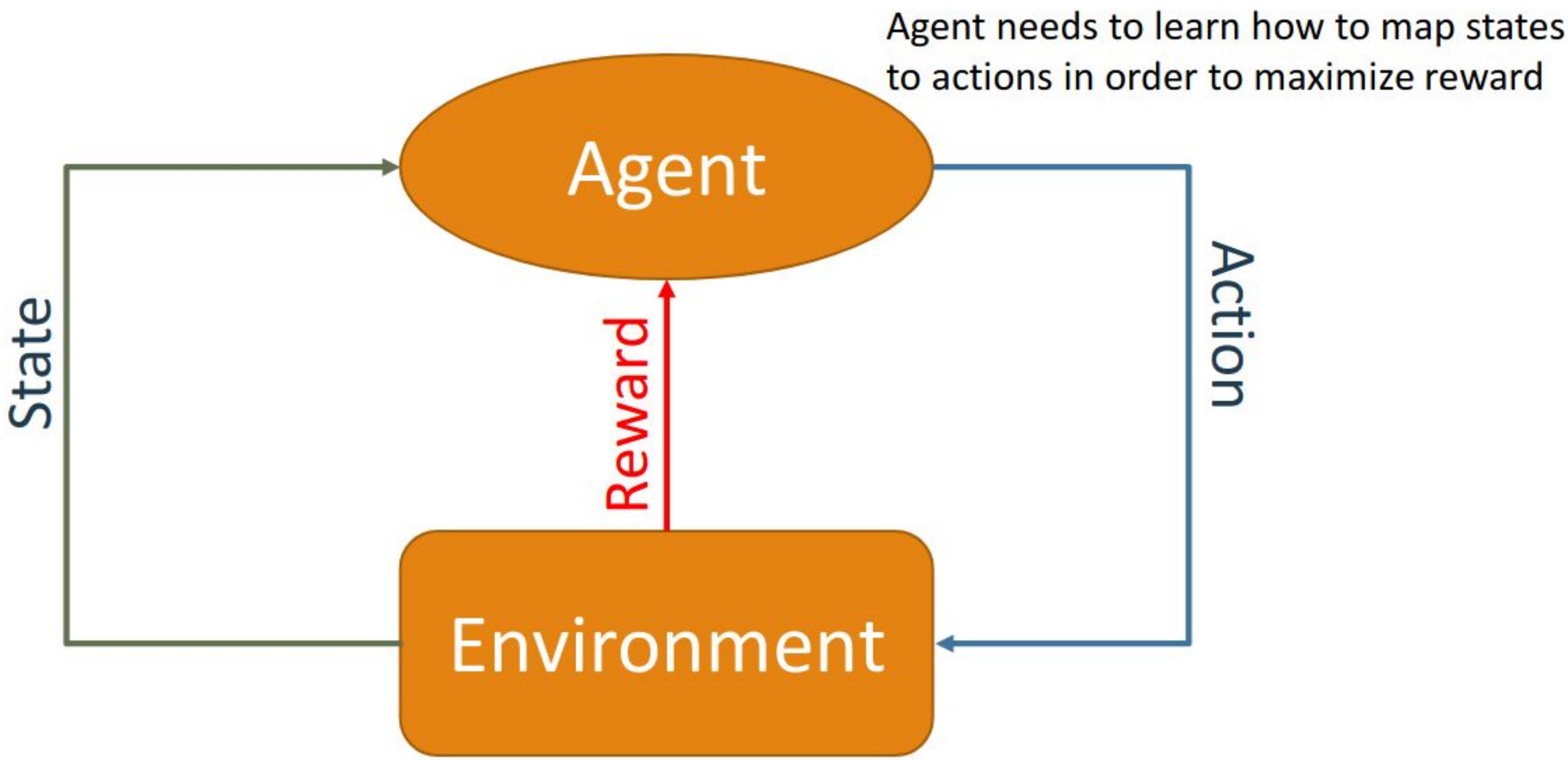
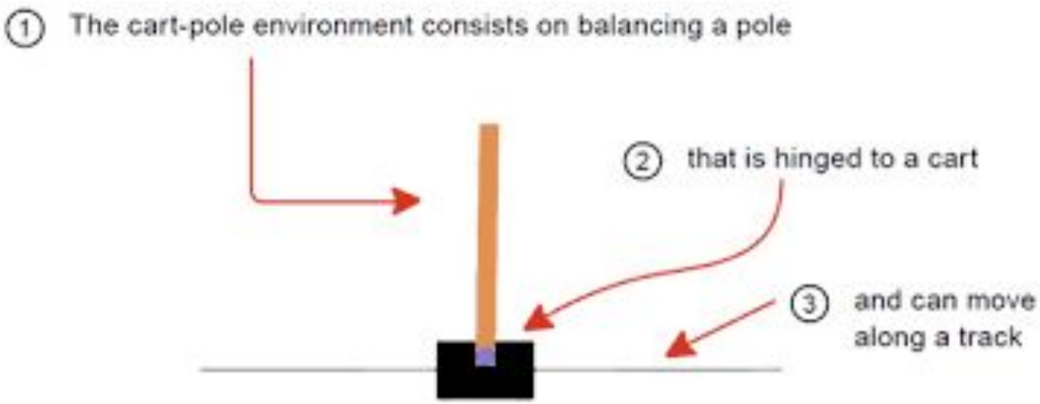
RL Overview



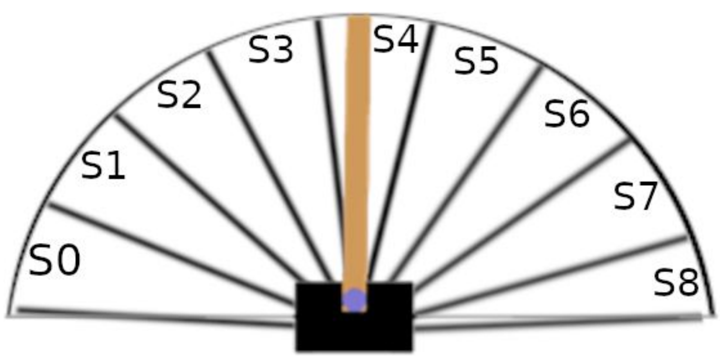
Code for the Reinforcement Learning program can be found here:
https://github.com/rfebbo/ReinforcementLearning_Cpp

Defining the Environment



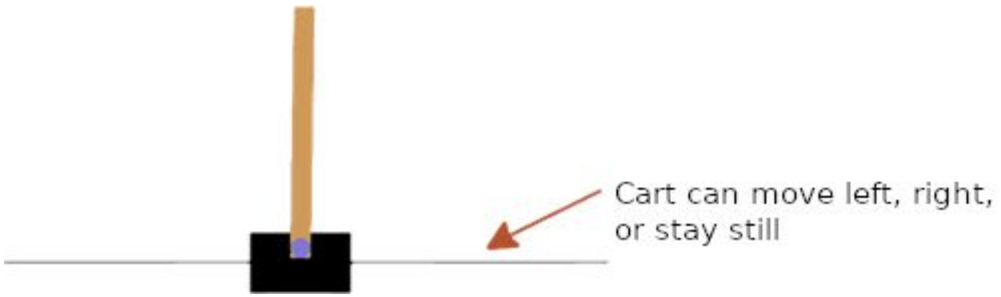
The pole is simulated using a set of equations derived from a free body diagram. https://coneural.org/florian/papers/05_cart_pole.pdf

Defining the State Space



- The agent is sent a state value from the environment.
- The state consists of the angle of the pole.
- The agent is only aware of the different descritized states which the pole can exist in. (The poles angle is not continuous from the agents point of view)
- A higher “resolution” can lead to better training results
- This discretization limitation can be eliminated with modern RL approaches such as DeepQ Learning

Defining the Action Space




```
void Env::step_action(int action) {
 double new_force;

 if (action == 0)
   new_force = -ep.input_force; //predefined constant
 else if (action == 1)
   new_force = 0;
 else if (action == 2)
   new_force = ep.input_force;
 else
   fprintf(stderr, "INVALID ACTION\n");

 this->step_force(new_force); // call function containing
                              // equations of motion to simulate
}
```

Defining the Reward

This is somewhat of an open question for this problem. Reward can be defined in many ways. Here are a few examples:

- The angular distance from the terminal state in which the pole exists
- 0 for all states and -1 for the terminal state

```
double body::get_R(double p) {
 switch (bp.r_type) {
   case R_Type::ENDS: {
     if (p >= bp.end_position_2 || p <= bp.end_position_1)
       return -1;
     return 0;
     break;
   }
   case R_Type::DISTANCE: {
     if (p == positions[mid_point])
       return 0;

     double distance = p - positions[mid_point];

     return -(distance * distance);
     break;
   }
   default:
     break;
 }

 fprintf(stderr, "INVALID R_Type\n");
 return 0;
}
```

Defining the Agent

The agents objective is to maximize rewards

At each timestep in the episode:

1. Agent receives a state value [0:8]
2. Agent uses Q table to lookup the best action to take (maxQ(s))
3. Agent sends action to environment
4. Environment simulates and sends the new state and a reward
5. Agent updates Q table for previous state based on:
   a. Reward
   b. Value at Q table for current state
   c. The action taken from step 2
6. Determine if the episode is over

```
double delta = p.reward_incentive * reward;
delta += p.discount * max_q;
delta -= Q[prev_state * num_actions + prev_action];
delta *= p.learning_rate;

Q[prev_state * num_actions + prev_action] += delta;
```

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}}$$

A Trained Q Table		**State**								
		S0	S1	S2	S3	S4	S5	S6	S7	S8
Actions	Left	0.0	0.1	0.1	0.2	0.2	0.6	0.7	0.8	0.9
	Nothing	0.1	0.1	0.2	0.2	0.6	0.2	0.2	0.1	0.1
	Right	0.9	0.8	0.7	0.6	0.2	0.2	0.1	0.1	0.0

https://en.wikipedia.org/wiki/Q-learning