```
function Q=Qlearning(M,options)
  nstates = 3^9;          % 状態数
  nactions = 9;       % 行動数
  results = zeros(M,1);   % 勝敗結果
  eM =1000;         % 評価を行うエピソード数

  % Q関数の初期化
  Q=zeros(nstates,nactions);

  for m=1:M
    rand('state',mod(m,eM))
    t = 1;
    state3 = zeros(1,9);

    while(1)
      % 状態, 報酬, ゲーム状況の観測
      state = encode(state3);

      %=====================
      % 政策の生成
      policy = zeros(1,nactions);

      switch(options.pmode)
        case 1 % greedy
          [v,a] = max(Q(state,:));
          policy(a) = 1;

        case 2 % e-greedy
          [v,a] = max(Q(state,:));
          policy = ones(1,nactions)*options.epsilon/nactions;
          policy(a) = 1-options.epsilon+options.epsilon/nactions;

        case 3 % softmax
          policy=exp(Q(state,:)/options.tau)/sum(exp(Q(state,:)./options.tau));
      end
      %=====================

      % 行動の選択および実行
      [action,reward,state3,fin] = action_train(policy,t,state3);

      %=====================
      % Q関数の更新 (Q学習)
      % 1ステップ前の状態, 行動のQ値を更新
      if t > 1
        Q(pstate,paction) = Q(pstate,paction) + options.alpha*(reward - Q(pstate,paction) + options.↙
gamma*max(Q(state,:)));
      end

      % ゲーム終了
      if(fin>0)
        results(m) = fin;
        break;
      end

      % 状態と行動の記録
      pstate = state;
      paction = action;

      t = t + 1;
    end

    if(mod(m,eM)==0)
      fprintf(1,'%d) Win=%d/%d, Draw=%d/%d, Lose=%d/%d¥n',m, size(find(results(m-eM+1:m)==2),1),eM,size(find(results(m-eM+1:m)==3),1),eM,size(find(results(m-eM+1:m)==1),1),eM);
    end

    fflush(stdout);
  end

  %  グラフの出力
  results2(results~=2)=0;
  results2(results==2)=1;
  res =reshape(results2,eM,M/eM);
  rate = sum(res)./eM;
  figure(3)
  clf
%  axes('FontSize',15,'LineWidth',2.0);
  games = eM:eM:M;
  g=plot(games,rate);
  set(g,'LineWidth',2);
  g=xlabel('ゲーム数');
  set(g,'FontSize',14);
  g=ylabel('勝率');
  set(g,'FontSize',14);
  axis([eM,M,0.4,1])
```