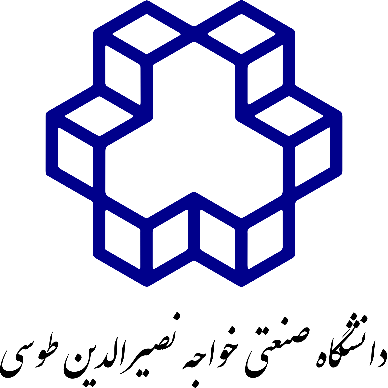
**به نام خدا**

****

**دانشکده مهندسی برق و کامپیوتر**

**دوره کارشناسي ارشد مهندسي برق**

**پروژه درس یادگیری تقویتی**

**توسط:**

**محمدرضا دیندارلو**

**40030824**

**استاد درس:**

**دکترخواسته**

**بهار 1402**

فهرست مطالب

[فهرست شکل ها 2](#_Toc140644220)

[سوال 4](#_Toc140644221)

[بخش اول 6](#_Toc140644222)

[بررسی نتایج Q-Learning 11](#_Toc140644223)

[بررسی نتایج SARSA 13](#_Toc140644224)

[بخش دوم 16](#_Toc140644225)

[بخش سوم 17](#_Toc140644226)

[بخش چهارم 18](#_Toc140644227)

# فهرست شکل­ها

[شکل 1: نتایج مربوط به بررسی Q-Learning با آموزش یک ملیون دوره 11](#_Toc140644261)

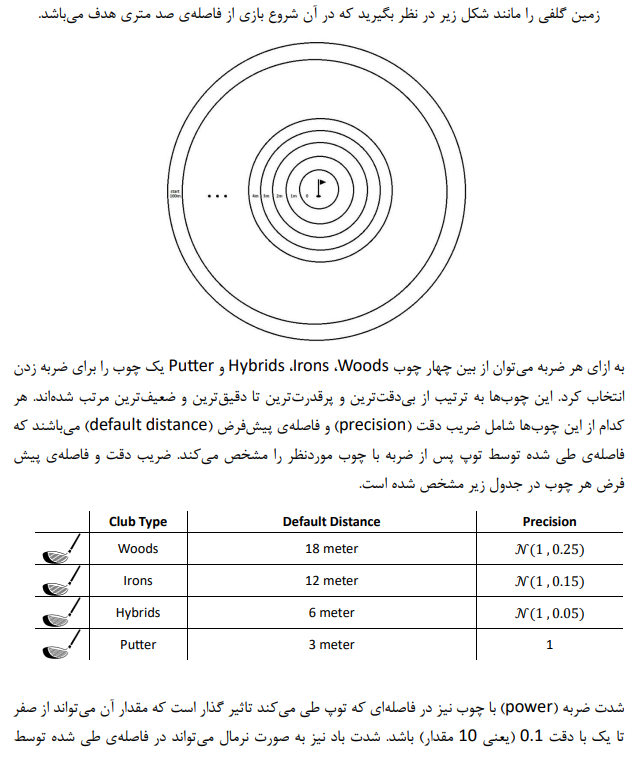
[شکل 2: پالیسی بهینه بدست آمده از الگوریتم Q-learning 12](#_Toc140644262)

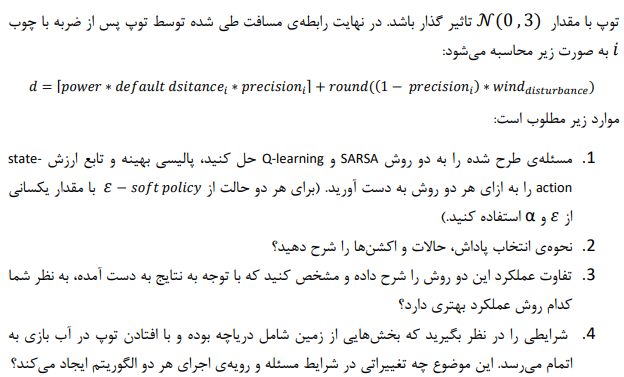
[شکل 3: الگوریتم SARSA با صد هزار دوره آموزش 14](#_Toc140644263)

[شکل 4: پالیسی بهینه بدست آمده با استفاده از الگوریتم SARSA 15](#_Toc140644264)

[شکل 5: نتیجه آموزش در حضور دریاچه 18](#_Toc140644265)

# سوال





# بخش اول

برای حل، مسئله را به چندین زیر بخش تبدیل می کنیم و توابع مورد نیاز هر بخش را توسعه می دهیم. اولین قسمت مربوط به محیط می باشد. تابع نوشته شده برای محیط بدین صورت می باشد که تمامی حالت ها، حالت فعلی عامل، هدف و اکشن فعلی را به عنوان ورودی در نظر می گیرد و با توجه به فضای محیط پاداش دریافتی از انتقال به حالت بعدی را به همراه حالت بعدی به عنوان خروجی ایجاد می کند. مطابق با مواردی که در سوال مطرح شده است، تعداد اکشن‌های مجاز چهل مورد می‌باشد. بر اساس انتخاب اکشن، فاصله‌ای که توپ طی می‌کند متفاوت خواهد بود و همچنین عامل باد نیز تأثیر متفاوتی دارد.

function [NewState , Reward] = Environment(AllStates , State , Action , Goal)

این تابع به صورت زیر پیاده سازی می شود:

old\_state = State(2) ;

power\_of\_action = mod(Action,10)\*0.1 ;

if(power\_of\_action == 0)

power\_of\_action = 1;

end

type\_of\_action = floor((Action-1)/10)+1 ;

% if type\_of\_action = 1 --> Putter selected default distance :3

% if type\_of\_action = 2 --> Hybrids selected default distance :6

% if type\_of\_action = 3 --> Irons selected default distance :12

% if type\_of\_action = 4 --> Woods selected default distance :18

برای پیاده سازی این منطق خواهیم داشت:

switch(type\_of\_action)

case 1

% Putter selected

default\_distance = 3;

precision = normrnd(1,0);

case 2

% Hybrids selected

default\_distance = 6;

precision = normrnd(1,0.05);

case 3

% Irons selected

default\_distance = 12;

precision = normrnd(1,0.15);

case 4

% Woods selected

default\_distance = 18;

precision = normrnd(1,0.25);

end

wind\_disturbance = normrnd(0,3);

بنابراین اکشن انتخاب شده میزان جابه جایی را به صورت زیر ایجاد می کند:

d = ceil(power\_of\_action\*default\_distance\*precision) + round((1-precision)\*wind\_disturbance);

حالت جدید به صورت زیر خواهد شد:

new\_state = abs( old\_state -d ) ;

همچنین پاداش های مورد نظر با توجه به اینکه حالت جدید حالت نهایی خواهد بود یا خیر تعیین خواهد شد:

if find(ismember(AllStates , NewState , 'row'))

if ismember(NewState , Goal , 'row')

Reward = +10;

else

Reward = -5;

end

else

Reward = -10 ;

NewState = [1 100];

end

در قسمت بعدی به بیان بدنه اصلی کد می پردازیم. باید زمین گلف را جوری شبیه سازی کنیم که بتوانیم در هر حالت فاصله تا مرکز را بدست آوریم. در این حالت خواهیم داشت:

game\_plane\_size = 101 ;

Plane = ones(1,game\_plane\_size) ;

imagesc(Plane) ;

می بینیم که حالت ها به صورت برداری در نظر گرفته می شود که هر آرایه از این بردار بیانگر فاصله توپ از مرکز زمین گلف می باشد. همچنین پر واضح است که هدف نهایی رساندن توپ به مرکز زمین است. بنابراین حالت ها را به صورت زیر تعیین می کنیم:

[R , C] = find(Plane == 1) ;

AllStates = [R' C'-1] ;

nStates = size(AllStates , 1) ;

فضای اکشن ها با توجه به اینکه از کدامین چوب با کدامین قدرت استفاده شود به صورت زیر خواهد شد:

AllActions = [0.3 0.6 1.2 1.8 ;...

0.6 1.2 2.4 3.6 ;...

0.9 1.8 3.6 5.4 ;...

1.2 2.4 4.8 7.2 ;...

1.5 3.0 6.0 9.0 ;...

1.8 3.6 7.2 10.8;...

2.1 4.2 8.4 12.6;...

2.4 4.8 9.6 14.4;...

2.7 5.4 10.8 16.2;...

3.0 6.0 12.0 18.0];

ستون ها بیانگر نوع چوب می باشد و ردیف ها بیانگر قدرت چوب انتخاب شده هستند. در ادامه فضای حالت اکشن را مشخص می کنیم تا در مراحل بعدی از آن استفاده کنیم.

nActions = size(AllActions , 1)\*size(AllActions , 2) ;

AllStateActionPairs = zeros(nStates\*nActions , 3);

for i = 1:nStates

for j = 1:size(AllActions , 2)

AllStateActionPairs(1+size(AllActions , 1)\*(4\*(i-1)+j-1):size(AllActions , 1)\*(4\*(i-1)+j) , :) = [repmat((AllStates(i , :)),size(AllActions , 1),1) , AllActions(1:size(AllActions , 1),j)];

end

end

بدین ترتیب فضای مسئله مشخص شده است و می توانیم از نتایج بالا در پیش برد پیاده سازی الگوریتم استفاده کنیم. برای پیاده سازی از پارامتر های زیر استفاده می کنیم:

nAllStateActionPairs = size(AllStateActionPairs , 1) ;

Counter = zeros(nAllStateActionPairs , 1);

Q = zeros(nAllStateActionPairs , 1);

gamma =0.99 ;

nEpisode = 1000000;

epsilon = 0.15 ;

Policy = randi(nActions , nStates , 1);

alpha = 0.1 ;

این پارامتر ها در در طی مراحل یادگیری دستخوش تغیرات مورد نیاز می کنیم. در ادامه بدنه اصلی کد های مربوط به Q-learning را بررسی می کنیم.

for e = 1:nEpisode

State = [1 100]; % Initial State

while 1

indexOfState = find(ismember(AllStates , State , 'row'));

% Choose an action based on the policy with epsilon-greedy exploration

randomNum = rand;

if randomNum < ((1 - epsilon) + (epsilon / nActions))

At = Policy(indexOfState); % Exploitation: choose action from policy

else

At = randi(nActions); % Exploration: choose a random action

end

[NewState , Reward] = Environment(AllStates , State , At , Goal) ;

if ismember(NewState , Goal , 'row')

break;

end

indexInPairs= find(ismember(AllStateActionPairs , [State At] , 'row'));

indexOfNextStateInPairs= find(ismember(AllStateActionPairs(: , [1 2]) , NewState , 'row'));

Q(indexInPairs) = Q(indexInPairs) + alpha\*(Reward+gamma\*max(Q(indexOfNextStateInPairs))-Q(indexInPairs));

State = NewState ;

Policy = PolicyImprovementUsingQ(Q , nActions) ;

if ismember(State , Goal , 'row')

break;

end

end

disp(['e = ' num2str(e)]) ;

end

در این بدنه اصلی حلقه ای برای تکرار مراحل یادگیری به تعداد اپیزود های تعریف شده مشاهده می کنیم. در این حلقه هر بار از نقطه شروع اولیه مشخص شده در صورت سوال کمک می گیریم و تا وقتی که به هدف برسیم اکشنی را انتخاب می کنیم و حالت ها را طی خواهیم کرد. در این مراحل هر بار پالیسی نیز بهبود می یابد که به صورت زیر تابعی برای آن در نظر گرفته شده است:

function Policy = PolicyImprovementUsingQ(Q , nActions)

nStates = size(Q , 1)/nActions ;

Policy = zeros(nStates , 1);

for s = 2:nStates

QQ = Q(1+nActions\*(s-1):nActions\*s) ;

idx = find(QQ == max(QQ)) ;

Policy(s) = idx(randi(numel(idx)));

end

end

همچنین مشخص است که از الگوریتم استفاده شده است. بدین ترتیب در این مرحله تابع ارزش حالت اکشن ها نیز تحت عنوان Q بدست می آید و هر مرحله به روز رسانی می شود و پالیسی بهینه نیز از آن استخراج می شود.

الگوریتم های خواسته شده در صورت سوال دو مورد Q-learning و SARSA هستند که پیاده سازی هر یک فقط در تعریف الگوریتم دارای اندکی تفاوت می باشد. ابتدا الگوریتم اول را پیاده سازی می کنیم و برای اپیزود های متفاوت این الگوریتم را تکرار می کنیم تا نتایج را بررسی می کنیم. و در نهایت دومین الگوریتم را پیاده سازی می کنیم. پیش از بررسی نتایج لازم است تابعی دیگر را ایجاد کنیم که در آن بتوانیم از پالیسی و تابع ارزش حالت اکشن ها استفاده کنیم و نتایج را بررسی کنیم.

## بررسی نتایج Q-Learning

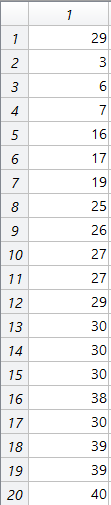
حال با آموزش عامل در محیط بیان شده با استفاده از نرم افزار متلب نتیجه به صورت زیر خواهد شد. این نتیجه برای آموزش عامل در محیط با یک ملیون دوره می باشد. مشاهده می کنیم که برای رسیدن به هدف هفت دوره نیاز داریم. البته این تعداد ممکن است که مقداری بیشتر و یا کمتر شود چرا که عامل بیرونی باد باعث ایجاد اغتشاش در اکشن های عامل می شود.



شکل 1: نتایج مربوط به بررسی Q-Learning با آموزش یک ملیون دوره

همچنین با تغیر نقطه شروع در کد های پیوست شده می توان نتایج را بیشتر بررسی نمود. مثلا با انتخاب نقطه شروع از جاهای دیگری از زمین ممکن است که تعداد گام های طی شده برای رسیدن به هدف تفاوت کند. با آموزش عامل با کمک این الگوریتم در Policy بدست آمده که در فایل های متلب ذخیره شده است مشاهده می کنیم که عامل ابتدا در حالت های دور از مرکز اکشن های آخر یعنی اکشن هایی که چوب انتخابی پر قدرت و همچنین کم دقت می باشد را انتخاب می کند و رفته رفته با نزدیک شدن به مرکز اکشن انتخابی اکشنی با دقت بالاتر و قدرت کمتر تبدیل می شود.

همچنین در شکل زیر می توانیم نتیجه نهایی پالیسی بهینه بدست آمده با استفاده از الگوریتم sarsa را بررسی کنیم. می بینیم که نتایج به همان صورتی می باشد که پیش­تر بیان کردیم. پالیسی نشان داده شده برای 20 حالت نهایی می باشد و مابقی حالت ها دارای اکشن 40 که همان چوب با قدرت و کم دقت می باشد هستند. این نتایج با انتظارات ما نسبت به اکشن های انتخابی توسط عامل همخوانی دارد.



شکل 2: پالیسی بهینه بدست آمده از الگوریتم Q-learning

همچنین برای دریافت بهتر موضوع عوامل اغتشاش صفر در نظر گرفته شده است و نتایج بررسی شده است که نتایج بدست آمده است و بیانگر همین موضوع می باشند.

## بررسی نتایج SARSA

در ادامه نتایج حاصل از آموزش الگوریتم sarsa را بررسی خواهیم کرد. این الکوریتم یک الگوریتم on-policy می باشد که با الگوریتم قبل تفاوتی در پیاده سازی دارد. همه توابع استفاده شده مطابق قبل می باشد. الگوریتم به صورت زیر پیاده سازی می شود:

while 1

indexOfState = find(ismember(AllStates, State, 'row'));

% Choose an action based on the policy with epsilon-greedy exploration

randomNum = rand;

if randomNum < ((1 - epsilon) + (epsilon / nActions))

At = Policy(indexOfState); % Exploitation: choose action from the policy

else

At = randi(nActions); % Exploration: choose a random action

end

[NewState, Reward] = Environment(AllStates, State, At, Goal);

if ismember(NewState, Goal, 'row')

break;

end

indexInPairs = find(ismember(AllStateActionPairs, [State, At], 'row'));

indexOfNextStateInPairs = find(ismember(AllStateActionPairs(:, [1, 2]), NewState, 'row'));

a\_prime = Policy(NewState(2)+1);

indexOfNextStateInPairs = indexOfNextStateInPairs(a\_prime);

Q(indexInPairs) = Q(indexInPairs) + alpha \* (Reward + gamma \* Q(indexOfNextStateInPairs) - Q(indexInPairs));

State = NewState;

Policy = PolicyImprovementUsingQ(Q, nActions);

end

disp(['Episode: ', num2str(e)]);

end

حال با همان پارامتر های قبلی این الگوریتم را اجرا می کنیم و نتایج را بررسی می کنیم. اجرای این الگوریتم برای صد هزار دوره انجام شده است.

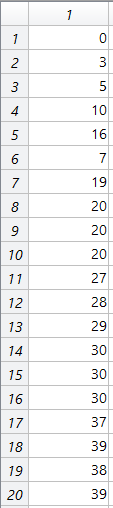
با آموزش عامل با کمک این الگوریتم در Policy بدست آمده که در فایل های متلب ذخیره شده است مشاهده می کنیم که عامل ابتدا در حالت های دور از مرکز اکشن های آخر یعنی اکشن هایی که چوب انتخابی پر قدرت و همچنین کم دقت می باشد را انتخاب می کند و رفته رفته با نزدیک شدن به مرکز اکشن انتخابی اکشنی با دقت بالاتر و قدرت کمتر تبدیل می شود. حال با آموزش انجام شده اگر از نقطه 100 شروع کنیم نتیجه به صورت زیر می شود:



شکل 3: الگوریتم SARSA با صد هزار دوره آموزش

همانطور که مشاهده می شود عامل با شش و یا هفت حرکت به هدف می رسد. برای مشاهده بهتر نتایج می توانید کد مربوطه را اجرا کنید و یا از نتایج ذخیره شده استفاده کنید و تنها انتهای کد را اجرا کنید تا اجرای الگوریتم به صورت گرافیکی را بررسی کنید.

همچنین در شکل زیر می توانیم نتیجه نهایی پالیسی بهینه بدست آمده با استفاده از الگوریتم sarsa را بررسی کنیم. می بینیم که نتایج به همان صورتی می باشد که پیش­تر بیان کردیم. پالیسی نشان داده شده برای 20 حالت نهایی می باشد و مابقی حالت ها دارای اکشن 40 که همان چوب با قدرت و کم دقت می باشد هستند. این نتایج با انتظارات ما نسبت به اکشن های انتخابی توسط عامل همخوانی دارد.



شکل 4: پالیسی بهینه بدست آمده با استفاده از الگوریتم SARSA

# بخش دوم

نحوه انتخاب اکشن ها، حالت ها و پاداش ها چگونه بوده است؟ در یادگیری تقویتی، عامل (agent) تلاش می‌کند در یک محیط (environment) خاص، اقداماتی یا اکشن‌ها را انجام داده و با مشاهده نتایج این اقدامات، بهترین استراتژی یا سیاست (policy) را برای دستیابی به هدف تعیین کند. برای انجام این کار، عامل به صورت تجربی در محیط عمل می‌کند و تلاش می‌کند با آزمون و خطا، اقدامات بهتری را انجام دهد تا به حالت‌های مطلوب بیانجامد. نحوه انتخاب اکشن‌ها، حالت‌ها و پاداش‌ها در یادگیری تقویتی به وسیله‌ی سه عنصر اساسی تعریف می‌شود:

حالت‌ها (States):

حالت‌ها یا موقعیت‌ها نشان‌دهنده وضعیت محیط در زمان‌های مختلف هستند. عامل به هر مرحله از زمان یک حالت را مشاهده می‌کند و اطلاعات موجود در حالت به عامل ارائه می‌شود تا براساس آن تصمیم‌گیری‌های خود را انجام دهد. در برخی موارد، حالت‌ها می‌توانند به صورت یک بردار ویژگی (feature vector) نمایش داده شوند که اطلاعات مهم محیط را دربردارند. در این سوال حالت موقعیت توپ در زمین می باشد.

اکشن‌ها (Actions):

اکشن‌ها نشان‌دهنده اقدامات یا حرکاتی هستند که عامل می‌تواند در هر حالت انجام دهد. انتخاب اکشن‌ها می‌تواند در نتیجه‌ی بررسی تاریخچه عملکرد عامل و به هدف‌های مطلوب هر مرحله اتخاذ شود. هدف عامل این است که با انتخاب بهترین اکشن‌ها به سمت دستیابی به اهدافش حرکت کند. در این سوال اکشن همان ضرباتی است که عامل به توپ وارد می کند که مطابق بخش اول بررسی شد و 40 اکشن در تمامی مسئله وجود دارد.

پاداش‌ها (Rewards):

پاداش‌ها نشان‌دهنده ارزشی است که عامل برای انجام هر اکشن در هر حالت از محیط دریافت می‌کند. هدف اصلی عامل در یادگیری تقویتی افزایش مجموع پاداش‌ها برای دستیابی به اهداف بلندمدت است. عامل در طول زمان یاد می‌گیرد که انجام کدام اکشن‌ها در کدام حالت‌ها منجر به پاداش بیشتری می‌شود و درنتیجه به بهترین سیاست یادگیری می‌پردازد. در این سوال دست یابی به هدف پاداش مثبت بزرگ و هر ضربه پاداشی منفی به عنوان جریمه برای عامل دارد که در این حالت عامل سعی در زودتر رسیدن به هدف را دارد.

ترکیب صحیح این سه عنصر (حالت‌ها، اکشن‌ها و پاداش‌ها) می‌تواند منجر به ایجاد روش‌ها و الگوریتم‌های یادگیری تقویتی موثری شود که عامل را به سمت دستیابی به هدف‌های مطلوب هدایت کند.

# بخش سوم

در یادگیری تقویتی، روش‌های On-Policy و Off-Policy دو رویکرد متفاوت برای یادگیری سیاست (policy) در محیط است. تفاوت اصلی بین این دو روش در این است که در روش On-Policy، سیاستی که عامل استفاده می‌کند، همان سیاستی است که برای یادگیری آن استفاده می‌شود، در حالی که در روش Off-Policy، سیاستی که عامل استفاده می‌کند، ممکن است با سیاستی که برای یادگیری آن استفاده می‌شود، متفاوت باشد.

* روش On-Policy:

در روش On-Policy، سیاستی که برای یادگیری استفاده می‌شود، همان سیاستی است که عامل در حال استفاده از آن در محیط است. عامل تجربه‌ها و پاداش‌ها را بر اساس سیاست فعلی خود جمع‌آوری می‌کند و براساس آن‌ها سیاست خود را به‌روزرسانی می‌کند. این روش به معنای "روش بر سیاست" است، زیرا عامل روی سیاست خود آموزش می‌بیند و تلاش می‌کند سیاست خود را بهبود بخشد. مثال‌هایی از روش‌های On-Policy شامل روش تخمین عملکرد Sarsa هستند.

* روش Off-Policy:

در روش Off-Policy، سیاستی که برای یادگیری استفاده می‌شود، ممکن است با سیاستی که عامل در حال استفاده از آن در محیط است، متفاوت باشد. عامل تجربه‌ها و پاداش‌ها را بر اساس سیاستی که در حال یادگیری آن استفاده می‌کند، جمع‌آوری می‌کند و براساس آن‌ها سیاست خود را به‌روزرسانی می‌کند. این روش به معنای "روش خارج از سیاست" است، زیرا عامل به صورت مستقل از سیاستی که در حال استفاده از آن است، یادگیری می‌کند. مثال‌هایی از روش‌های Off-Policy شامل روش Q-Learning هستند.

تفاوت اصلی بین روش On-Policy و Off-Policy در این است که در روش On-Policy، عامل بر روی سیاست خود آموزش می‌بیند و تلاش می‌کند آن را بهبود بخشد، در حالی که در روش Off-Policy، عامل با استفاده از داده‌های جمع‌آوری شده بر اساس سیاستی متفاوت، یادگیری می‌کند و سیاست خود را بهبود می‌دهد. هر یک از این روش‌ها مزایا و کاربردهای خاص خود را دارند و بسته به مسئله مورد نظر، انتخاب مناسبی است.

نتایج بدست آمده نشان می دهد که هر دو الگوریتم همگرا شده اند و می توانند با 6 الی 8 قدم به هدف برسند. آنچه در این دو الگوریتم تغییر کرده است ایجاد یک عبارت Max در الگوریتم ها می باشد که این موضوع باعث کندتر اجرا شدن الگوریتم Q-Learning شده است ولی واضح بود که این الگوریتم زودتر و با دوره های کمتری به هدف نهایی میل می کند. به طور کلی پیاده سازی الگوریتم Q-Learning در مسائلی این شکلی مناسب تر است.

# بخش چهارم

برای اینکه این شرایط را مهیا سازیم کافیست در فضای حالت ها قسمت هایی را به طریقی در نظر بگیریم که ورود به آنها پاداشی بزرگ و منفی را به دنبال داشته باشد. در این حالت در آموزش، عامل از ورود به این حالت ها که همان دریاچه ها هستند ممانعت می کند. کافیست فضای دریاچه را در کد مشخص کرده و به فضای غیر قابل ورود انتساب دهیم.

Plane(1,10:14) = 0;

Plane(1,80:90) = 0;

حال نتایج در این حالت به صورت زیر خواهد شد. تعداد دوره های آموزش در این قسمت یک ملیون دوره بوده است.



شکل 5: نتیجه آموزش در حضور دریاچه

می بینیم که در مناطق آبی رنگ عامل وارد نمی شود. (برای بررسی دیداری کد های مربوطه اجرا شود)

به صورت کلی در رویه اجرای دو الگوریتم هیچ تفاوتی ایجاد نمی کند و فقط در پاداشی که از محیط نسبت به حالت رفته شده بر می گردد تفاوت ایجاد می شود و مابقی الگوریتم همان قبلی می باشد.