پروژه ی کوریدور

علی فتاحی (شهریار)

صدف ترابی اردکانی

تیر ماه 1400

دانشگاه علم و صنعت

مبانی برنامه نویسی

استاد رفیعی

مهدی مصطفوی

Quoridor

فهرست

بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

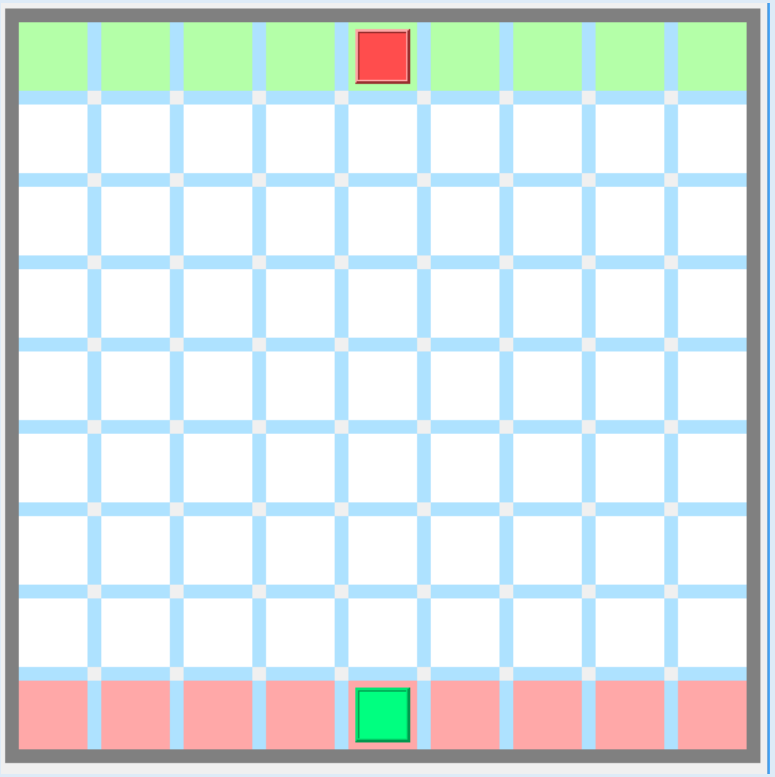
قانون کلی : هر مهره در طول بازی ، باید به حداقل یکی از خانه های انتهایی رو به روی خود دسترسی داشته باشد.

بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

برای مثال مهره ی قرمز ، در تمام طول بازی باید به حداقل یکی از خانه های قرمز رو به روی خود دسترسی داشته باشد.

هم چنین مهره ی سبز نیز باید به حداقل یکی از خانه های سبز رنگ رو به روی خود دسترسی داشته باشد.
(مسیری وجودداشته باشد که مهره سبز رنگ از طریق ان ، به خانه سبز برسد.)



بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

فرض کنید میخواهیم بررسی کنیم که آیا مهره ی قرمز محصور شده است یا نه ؟ (یا به عبارتی به خانه های قرمز رو به روی خود دسترسی دارد یا نه ؟)
برای این کار از الگوریتم زیر کمک میگیریم :

۱- ابتدا ارزش همه ی خانه ها را برابر با مقدار (۱-) میگذاریم ، به جز خانه ای که در ان مهره قرمز وجود دارد.
۲- ارزش خانه ای که در ان مهره ی قرمز وجود دارد برابر (0) میشود.
۳- متغییر step را با مقدار اولیه ی صفر تعریف میکنیم :

۴- خانه ای را که ارزش آن برابر با مقدار step است را پیدا کن :
۴-۱- اگر (در بالای این خانه دیوار نبود) و (ارزش خانه ی بالایی برابر (۱-) بود) : ارزش خانه ی بالایی را برابر با step+1 قرار بده.
۴-۲- اگر (در سمت راست این خانه دیوار نبود) و (ارزش خانه ی سمت راست (۱-) بود) : ارزش خانه ی سمت راست را برابر با step+1 قرار بده.
۴-۳- اگر (در سمت چپ این خانه دیوار نبود) و (ارزش خانه ی سمت چپ برابر (۱-) بود) : ارزش خانه ی سمت چپ را برابر با step+1 قرار بده.
۴-۴- اگر (در پایین این خانه دیوار نبود) و (ارزش خانه ی پایینی برابر (۱-) بود) : ارزش خانه ی پایینی را برابر step+1 قرار بده.

۵- اگر تمام خانه ها ارزش گذاری شدند : تمام
در غیر این صورت به مقدار step یک عدد اضافه کن و به خط شماره ۴ برگرد.

Quoridor

بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

-1	-1	-1	-1	0	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

همه ی خانه ها ارزش منفی یک میگیرند.
خانه ای که مهره ی قرمز در ان است ، ارزش صفر میگیرد.

بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

خانه هایی که مقدار ان ها step یا همان یک است شناسایی و مقدار خانه های اطراف ها میشود step+1 یا مقدار ۲ :

2	1	0	1	2
-1	2	1	2	-1
-1	-1	2	-1	-1

مقدار step برابر صفر است ، خانه ای که مهره ی قرمز در ان است شناسایی میشود و خانه های اطراف ان مقدار Step+1 که همان ۱ است ارزش گذاری میشود :

1	0	1
-1	1	-1

Step = 1

Step = Step + 1

روند ارزش گذاری را تا جایی که همه ی خانه ها ارزش گذاری شوند ، ادامه میدهیم.

Quoridor

بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

ارزش هر خانه بعد از ارزش گذاری :
(توجه کنید که در این صفحه هیچ دیواری به جز دیوار های مرزی وجود ندارد.)

4	3	2	1	0	1	2	3	4
5	4	3	2	1	2	3	4	5
6	5	4	3	2	3	4	5	6
7	6	5	4	3	4	5	6	7
8	7	6	5	4	5	6	7	8
9	8	7	6	5	6	7	8	9
10	9	8	7	6	7	8	9	10
11	10	9	8	7	8	9	10	11
12	11	10	9	8	9	10	11	12

Quoridor

بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

4	3	2	1	0	1	2	3	4
5	4	3	2	1	2	3	4	5
6	5	4	3	2	3	4	5	6
7	6	5	4	3	4	5	6	7
8	7	6	5	4	5	6	7	8
9	8	7	6	5	6	7	8	9
10	9	8	7	6	7	8	9	10
11	10	9	8	7	8	9	10	11
12	11	10	9	8	9	10	11	12

نتیجه

اگر ارزش حداقل یک خانه از خانه های انتهایی (خانه هایی که ارزششان با رنگ قرمز مشخص شده) ارزشی غیر از ۱- بود ، یعنی مهره مسیری برای حرکت به ان خانه دارد پس مهره محاصره نشده.

برای فهم بهتر در صفحه بعد دو مثال بررسی میکنیم.

بررسی الگوریتم ها

بخش ۱ : الگوریتم محاصره شدن

مثال

10	3	2	1	0	1	14	13	14
9	8	7	2	1	2	15	12	13
8	7	6	3	4	17	16	11	12
9	8	5	4	5	8	9	10	11
12	11	10	9	6	7	8	9	10
11	10	9	8	7	8	9	10	11
12	11	10	9	8	9	10	11	12
13	12	11	10	9	10	11	12	13
14	13	12	11	10	11	12	13	14

در این مثال به دلیل محصور نبودن مهره قرمز ، ارزش خانه های انتهایی از منفی یک تغییر کرده.

10	3	2	1	0	1	-1	-1	-1
9	8	7	2	1	2	-1	-1	-1
8	7	6	3	4	-1	-1	-1	-1
9	8	5	4	5	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

در این مثال به دلیل محصور بودن مهره قرمز ، ارزش همه ی خانه های انتهایی منفی یک مانده.

Quoridor

بررسی الگوریتم ها

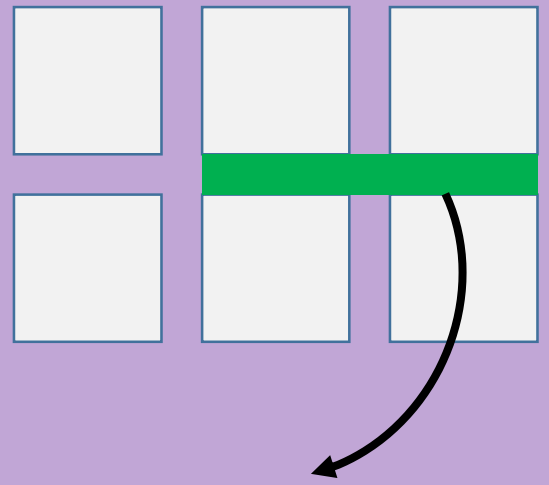
بخش ۲ : الگوریتم دیوارکشی

قانون کلی : هر بازیکن ۱۰ دیوار در کل بازی دارد. هر دیوار باید فقط دو خانه را درگیر کند.روی دیوارها نمیتوان دیوار جدید گذاشت.

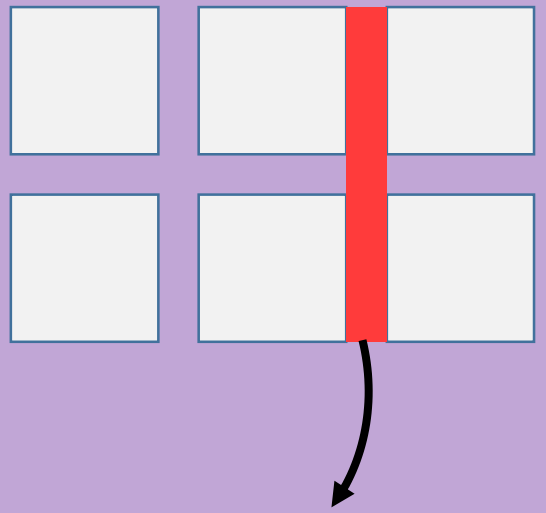
Quoridor

بررسی الگوریتم ها

بخش ۲ : الگوریتم دیوارکشی



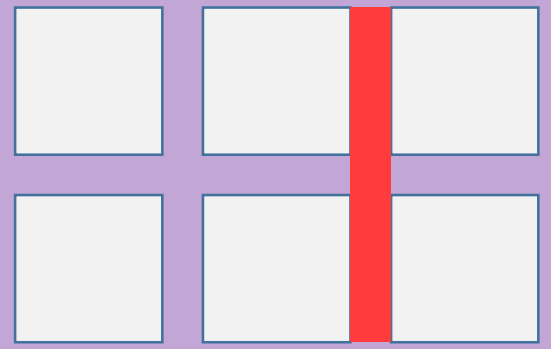
دیوار سبز دو خانه ی ستونی را درگیر کرده

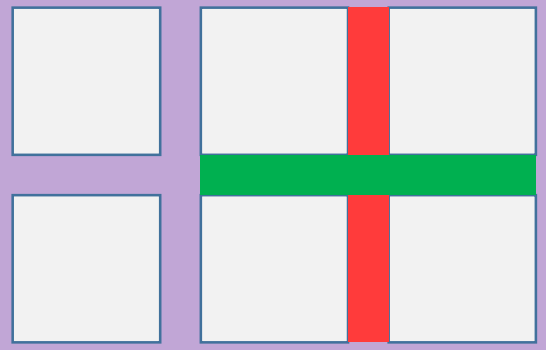


دیوار قرمز دو خانه ی سطری را درگیر کرده

بررسی الگوریتم ها

بخش ۲ : الگوریتم دیوارکشی





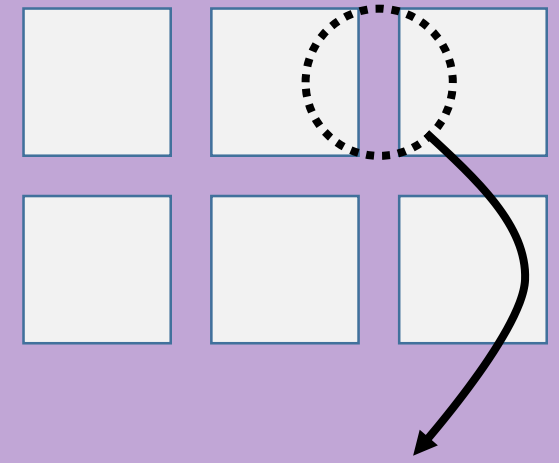
1 اگر در حالت اول ، چنین دیواری قرار داده شد

2 در حالت دوم نمیتوان روی دیوار قرمز ، دیوار سبز قرار داد

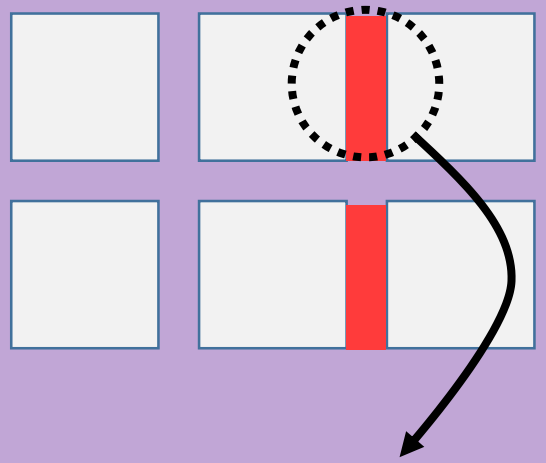
بررسی الگوریتم ها

بخش ۲ : الگوریتم دیوارکشی

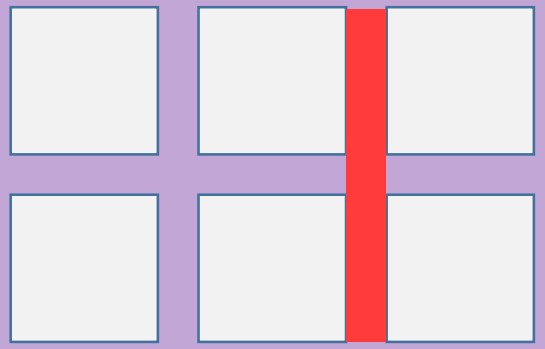
باید دقت داشته باشیم هر دیوارچینی شامل قرار دادن ۲ دیوار در برنامه است.



با کلیک بر روی این قسمت ، یک دیوار در این قسمت و یک دیوار در زیر ان قرار داده میشود (در واقع این قسمت ها Button هستند که با کلیک بر روی ان ، رنگ ان به قرمز و یا سبز تغییر میکنند.)



این دیوار که در برنامه یک Button است افزایش طول میابد.



دیوار نهایی

بررسی الگوریتم ها

بخش ۲ : الگوریتم دیوارکشی

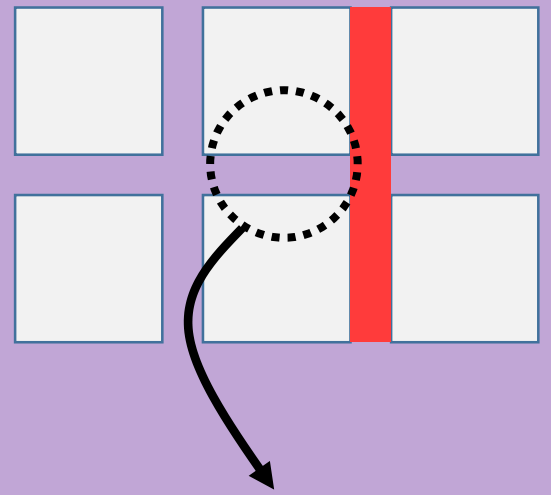
برای جلوگیری از قرار دادن یک دیوار بر روی دیوار دیگر به گونه زیر عمل میکنیم :

- اگر قرار است دیوار افقی جدیدی قرار دهیم ، در صورتی مجاز است :
  دو دیوار عمودی کناری (بالا و پایین) به رنگ قرمز یا سبز نباشند.

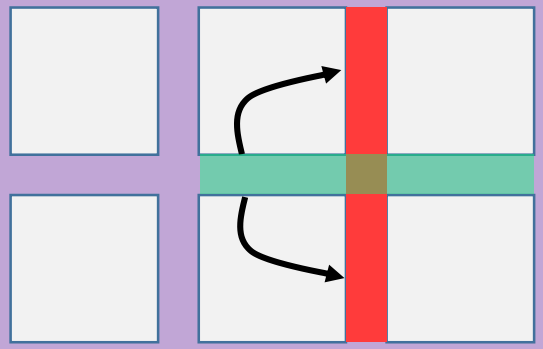
- اگر قرار است دیوار عمودی جدیدی قرار دهیم ، در صورتی مجاز است :
  دو دیوار افقی زیر (در سمت چپ و راست) به رنگ قرمز یا سبز نباشند.

بررسی الگوریتم ها

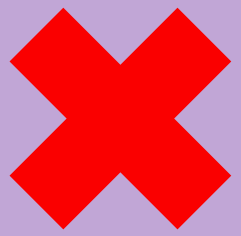
بخش ۲ : الگوریتم دیوارکشی



برای مثال فرض کنید میخواهیم در این قسمت یک دیوار افقی جدید قرار دهیم



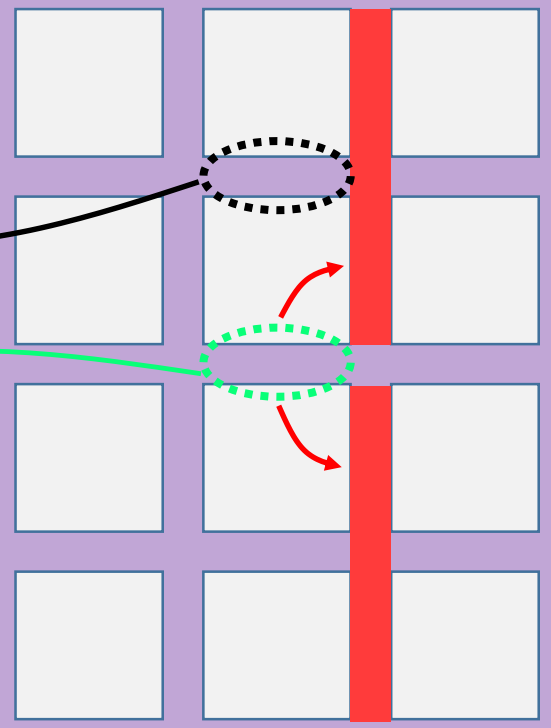
چون دو دیوار عمودی سمت راست (بالا و پایین) به رنگ قرمز هستند پس نمیتوان در این محل دیوار افقی قرار داد.



بررسی الگوریتم ها

بخش ۲ : الگوریتم دیوارکشی

اما روش قبلی دارای نقصان است. دو حالت زیر را در نظر بگیرید :



از نظر قوانین بازی در این محل نمیتوان دیوار افقی گذاشت

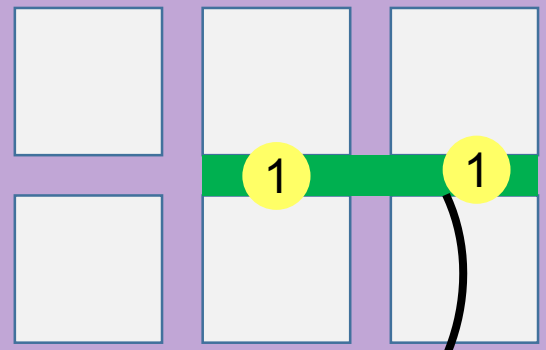
اما از نظر قوانین بازی در این محل میتوان دیوار افقی گذاشت. ولی در الگوریتم چون گفته ایم که اگر دیوار های عمودی کناری (در بالا و پایین) قرمز یا سبز بودند ، دیوار گذاشتن ممنوع است پس در الگوریتم قبلی نمیتوان در این محل دیوار گذاشت.

بررسی الگوریتم ها

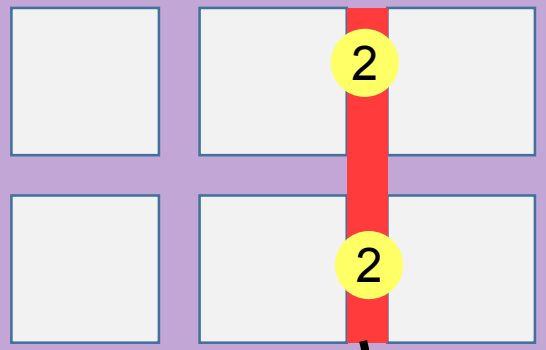
بخش ۲ : الگوریتم دیوارکشی

برای رفع این مشکل به تمام دو Button هایی که در مجموع تشکیل یک دیوار داده اند ، ارزش یکسان میدهیم.

برای مثال :



دو Button سبز رنگ که در مجموع یک دیوار افقی سبز رنگ تشکیل داده اند. ارزش هر دو باتن برابر مقدار ۱ میباشد.



دو Button قرمز رنگ که در مجموع یک دیوار عمودی قرمز رنگ تشکیل داده اند. ارزش هر دو باتن برابر مقدار ۲ میباشد.

بررسی الگوریتم ها

بخش ۲ : الگوریتم دیوارکشی

حالا الگوریتم را به گونه ی زیر اصلاح میکنیم (قسمت رنگی ، قسمت اضافه شده به الگوریتم قبلی است) :

- اگر قرار است دیوار افقی جدیدی قرار دهیم ، در صورتی مجاز است که :
دو دیوار عمودی کناری (بالا و پایین) به رنگ قرمز یا سبز نباشند یا اگر قرمز و سبز رنگ هستند دارای ارزش یکسان نباشند.

- اگر قرار است دیوار عمودی جدیدی قرار دهیم ، در صورتی مجاز است که :
دو دیوار افقی زیر (در سمت چپ و راست) به رنگ قرمز یا سبز نباشند یا اگر قرمز و سبز رنگ هستند دارای ارزش یکسان نباشند.

Quoridor

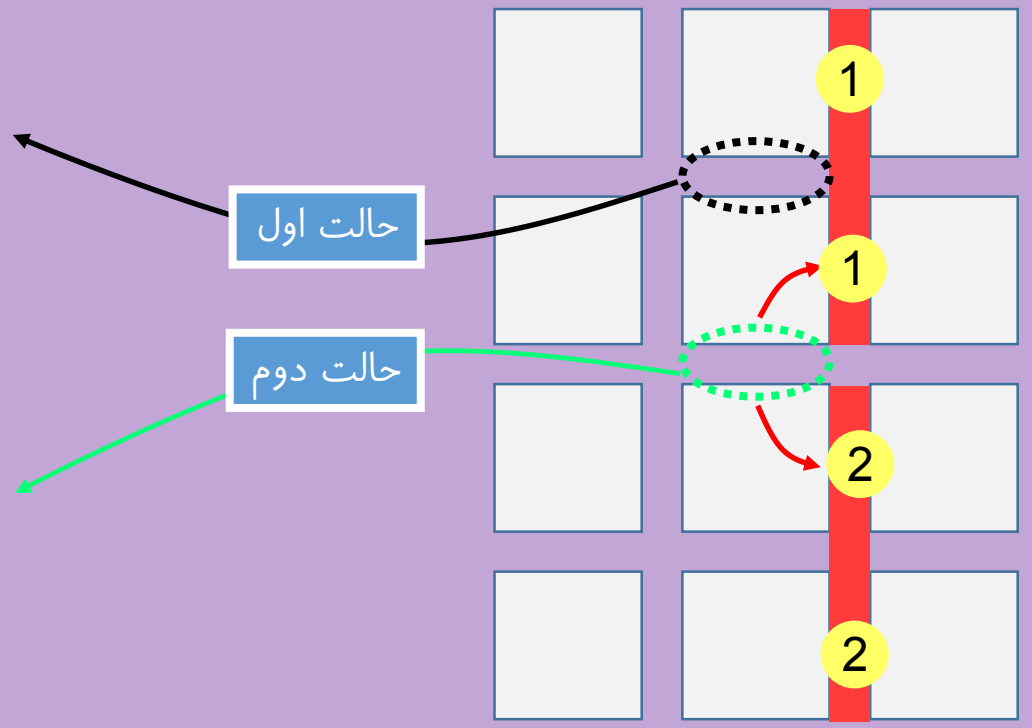
بررسی الگوریتم ها

بخش ۲ : الگوریتم دیوارکشی

حالا در الگوریتم جدید ، دو حالت زیر را در نظر بگیرید :

در این حالت چون دو دیوار عمودی کنار این محل (بالا و پایین) قرمز رنگ هستند و هم چنین دارای ارزش یکسان هستند(۱) پس در این محل گذاشتن دیوار افقی ممنوع است. ☹

اما در این حالت با اینکه دیوار های عمودی کناری این محل (بالا و پایین) قرمز رنگ هستند ولی دارای ارزش یکسان نیستند (۱ و ۲) برای همین در این محل گذاشتن دیوار افقی مجاز است. ☺



Quoridor

بررسی الگوریتم ها

بخش ۳ : الگوریتم حرکت مهره ها

قانون کلی : هر بازیکن میتواند تنها به خانه های اطراف خود در صورت باز بودن مسیر ، یک حرکت داشته باشد.

تبصره : اگر بازیکن حریف در خانه ی مجاور مهره قرار داشت ، مهره میتواند از روی مهره حریف پرش کند(دو خانه حرکت کند)

بررسی الگوریتم ها

با هر بار کلیک بر روی مهره که یک Button است ، اگر نوبت بازیکن باشد ، الگوریتم زیر اجرا میشود :

۱- خانه ای را که مختصات ان با مختصات مهره برابر است را پیدا کن : (x , y)

۱-۱- اگر بالای این خانه دیوار نیست ، خانه به مختصات (x , y+1) را به رنگ زرد در بیاور.
۱-۲- اگر سمت راست این خانه دیوار نیست ، خانه به مختصات (x+1 , y) را به رنگ زرد در بیاور.
۱-۳- اگر سمت چپ این خانه دیوار نیست ، خانه به مختصات (x-1 , y) را به رنگ زرد در بیاور.
۱-۴- اگر پایین این خانه دیوار نیست ، خانه به مختصات (x , y-1) را به رنگ زرد در بیاور.

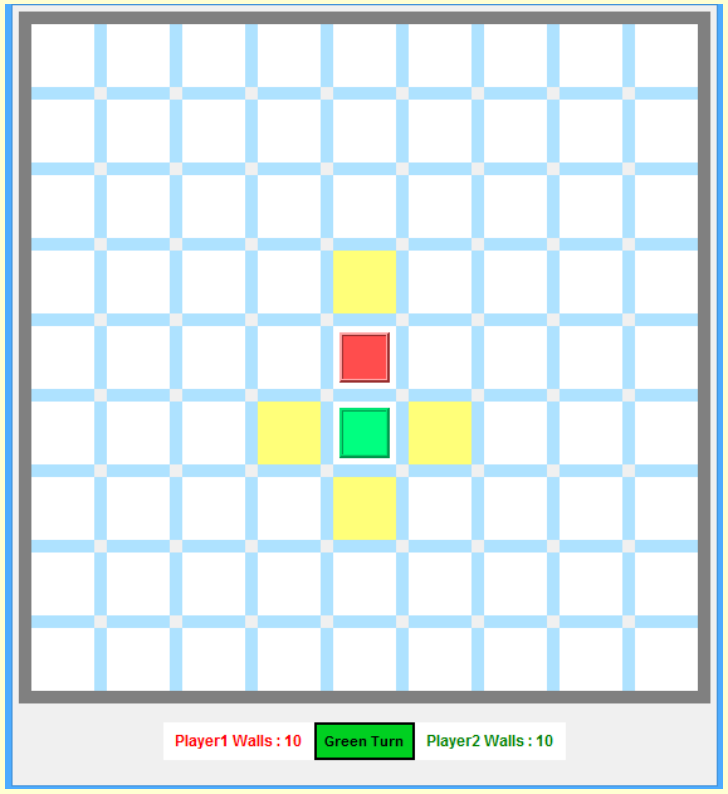
۲- سپس با کلیک بر روی یکی از خانه های زرد (خانه های مجاز برای حرکت) مهره با تغییر مختصات به سمت ان حرکت میکند.

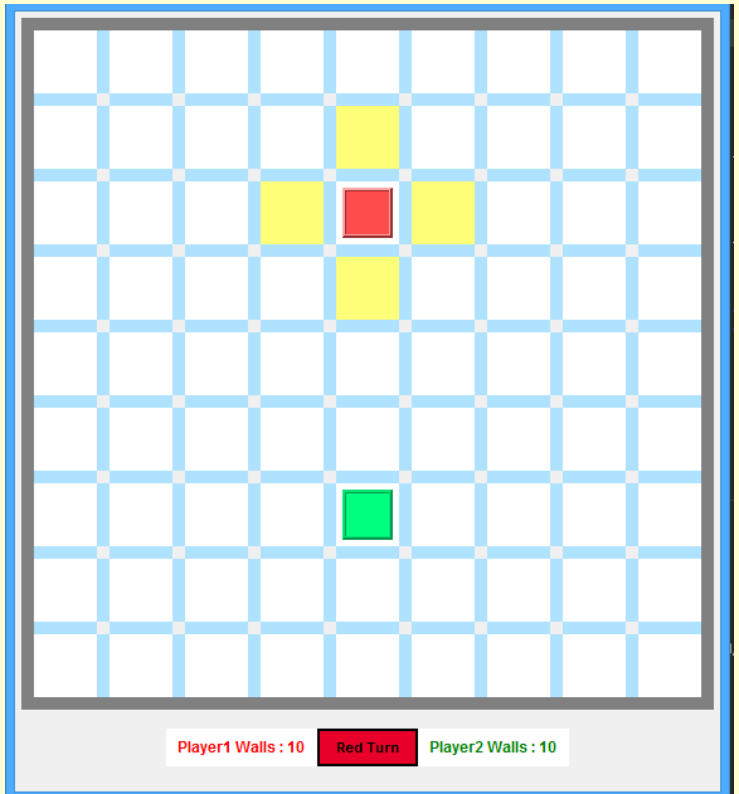
بررسی الگوریتم ها

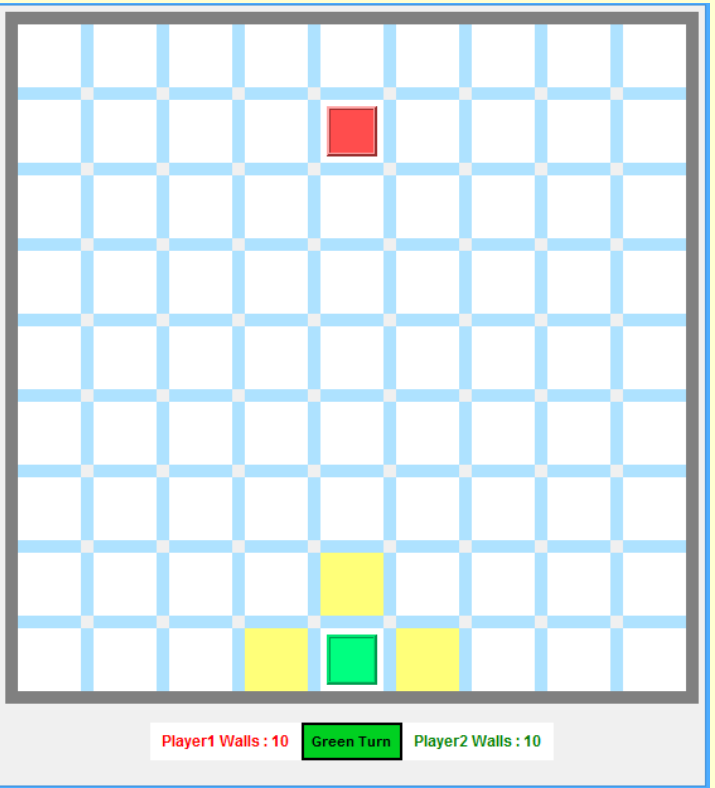
بخش ۳ : الگوریتم حرکت مهره ها

البته در الگوریتم قبلی باید چک کنیم که اگر خانه ای که قرار است زرد شود ، مهره حریف در ان قرار داشته باشد ، خانه بقلی ان به رنگ زرد در اید و نه همان خانه.

نمونه :







بخش دوم: آنالیز کد

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

assign_wall_to_block()
Line 103 To 121

clear_square_color() : 127 To 130
Reset_vblock() : 131 To 134

is_valid_wall(*x , y , typew*)
Line 135 To 240

creat_wall(*arg1 , arg2 , x , y , typew*)
Line 242 To 320

set_yellow_squarGoal(*arg,p*)
Line 321 To 410

move(*arg , y , x*)
Line 413 To 471

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

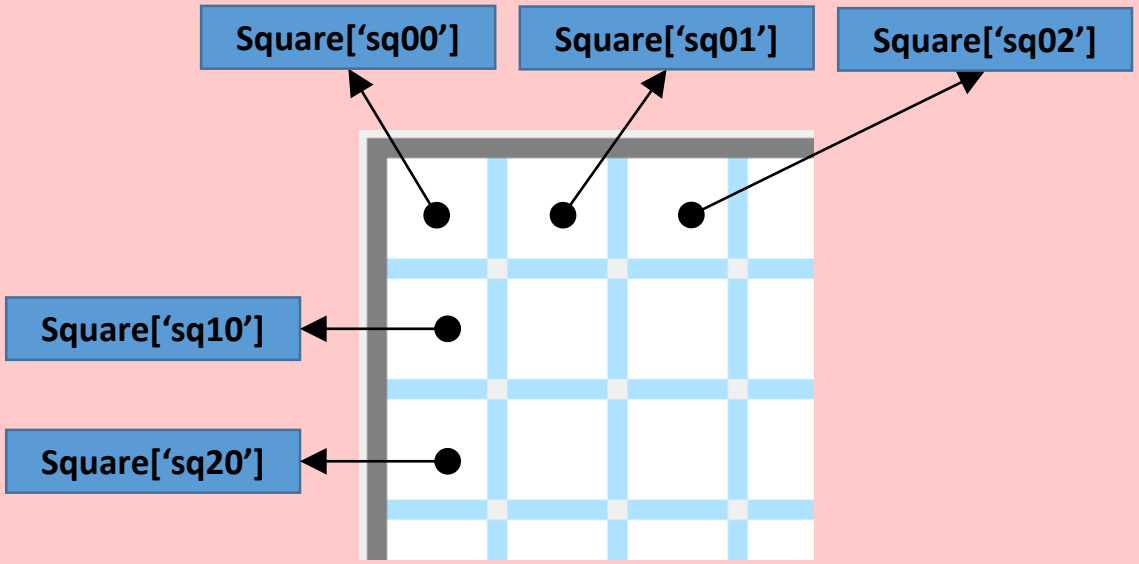
```
1   # \
2   import tkinter as tk
3   from tkinter import messagebox
4
5   Quoridor = tk.Tk()
6   Quoridor.geometry('560x620+300+0')
7   Quoridor.title("Quoridor Game")
8   Quoridor.resizable(width=False, height=False)
9
```

۵ : تعریف یک پنجره جدید با نام Quoridor
۶ : تنظیم ابعاد این پنجره و مختصات شروع پنجره
۷ : تیتر پنجره Quoridor Game تعریف میشود
۸ : ابعاد بازی غیر قابل تغییر تنظیم میشود.

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

```
12  Square = dict()
13  for i in range(9) :
14      for j in range(9) :
15          Square[f'sq{i}{j}'] = tk.Button(Quoridor , bg = "white" , bd = 0)
16          Square[f'sq{i}{j}'].place(height = 50 , width = 50 , y = (i+1)*10 + 50*i + 5 , x = (j+1) * 10 + j*50 + 5)
17          Square[f'sq{i}{j}'].config(command = lambda arg = Square[f'sq{i}{j}'] , y = i , x = j : move(arg,y,x))
18
19
```



در این قسمت ، تمام خانه های بازی ، یک باتن ۵۰*۵۰ در قالب یک دیکشنری به نام Square تعریف میشوند. در قسمت Config این خانه ها ، تنظیم میشود که وقتی کلیک شدند ، تابع ()move اجرا شود.

تابع move سه ورودی دارد :

X : مختصات x باتنی که روی ان کلیک شده

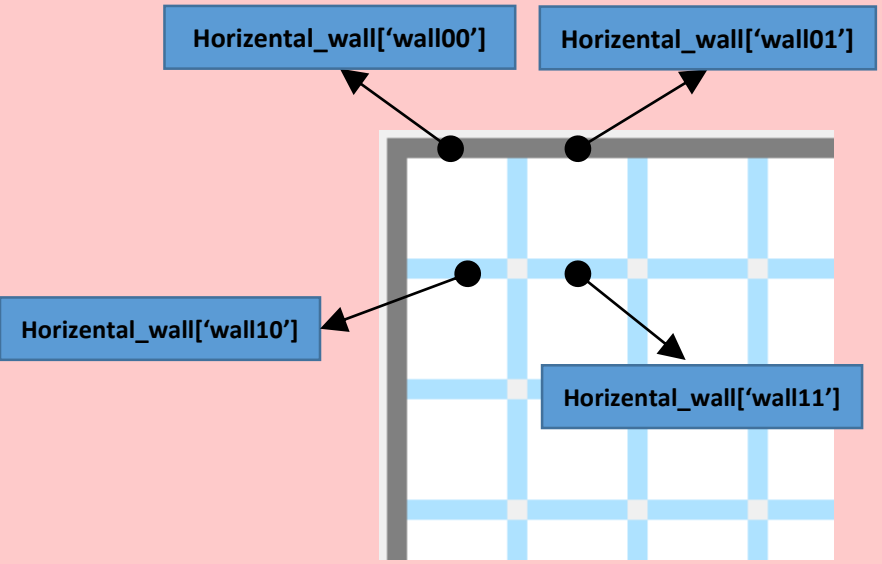
Y : مختصات y باتنی که روی ان کلیک شده

Arg : باتنی که روی ان کلیک شده در قالب یک شیئ به تابع پاس داده میشود

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

```
20
21  Horizental_wall = dict()
22  for i in range(10):
23      for j in range(9) :
24          if i == 0 or i == 9 :
25              Horizental_wall[f'wall{i}{j}'] = tk.Button(Quoridor , background = "gray" , bd = 0)
26              Horizental_wall[f'wall{i}{j}'].place(height = 10 , width = 60 , x = j * 50 +(j+1)*10 + 5, y = i*50 + i * 10 +5)
27          else :
28              Horizental_wall[f'wall{i}{j}'] = tk.Button(Quoridor , background = "#aee2ff" , bd= 0)
29
30              Horizental_wall[f'wall{i}{j}'].place(height = 10 , width = 50 , x = j * 50 +(j+1)*10 + 5, y = i*50 + i * 10 +5)
31          Horizental_wall[f'id{i}{j}'] = -1
32  Horizental_wall[f'wall00'].place(height = 10 , width = 70 , x = 5, y = 5)
33
```

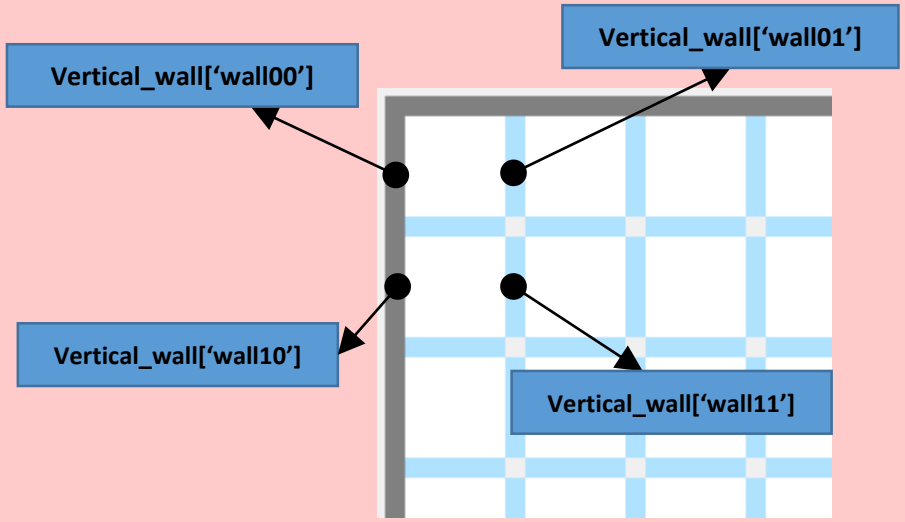


در این قسمت ، تمام دیوار های افقی در قالب دیکشنری با نام Horizental_wall ترسیم میشود. اگر i= 0 یا i= 9 یعنی دیوار های مرزی که به رنگ gray و بقیه دیوار ها به رنگ آبی کم رنگ ترسیم میشوند.
هم چنین Horizental_wall[id{i}{j}] با ارزش ابتدایی ۱- برای هر دیوار تعریف میشود. از این key برای الگوریتم دیوار چینی استفاده میشود.

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

```
36
37  Vertical_wall = dict()
38  for i in range(9) :
39      for j in range(10) :
40          if j == 0 or j == 9 :
41              Vertical_wall[f'wall{i}{j}'] = tk.Button(Quoridor , background = "gray" , bd = 0)
42              Vertical_wall[f'wall{i}{j}'].place(height = 60 , width = 10 , x = j * 50 + (j)*10 + 5, y = i*50+(i+1)*10 + 5)
43          else :
44              Vertical_wall[f'wall{i}{j}'] = tk.Button(Quoridor , background = "#aee2ff" , bd =0)
45
46
47              Vertical_wall[f'wall{i}{j}'].place(height = 50 , width = 10 , x = j * 50 + (j)*10 + 5, y = i*50+(i+1)*10 + 5)
48          Vertical_wall[f'id{i}{j}'] = -1
49
50
```



در این قسمت ، تمام دیوار های عمودی در قالب دیکشنری با نام vertical_wall ترسیم میشود. اگر i= 0 یا i= 9 یعنی دیوار های مرزی که به رنگ gray و بقیه دیوار ها به رنگ آبی کم رنگ ترسیم میشوند.
هم چنین vertical_wall[id{i}{j}] با ارزش ابتدایی ۱- برای هر دیوار تعریف میشود. از این key برای الگوریتم دیوار چینی استفاده میشود.

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

```
53
54  for i in range(10) :
55      for j in range(9) :
56          if j < 8 :
57              Horizental_wall[f'wall{i}{j}'].config(command = lambda arg1 = Horizental_wall[f'wall{i}{j}'] , arg2 = Horizental_wall[f'wall{i}{j+1}'] , x = j , y = i , typew = 'h' : creat_wall(arg1,arg2,x,y,typew))
58  for i in range(9) :
59      for j in range(10) :
60          if i < 8 :
61              Vertical_wall[f'wall{i}{j}'].config(command = lambda arg1 = Vertical_wall[f'wall{i}{j}'] , arg2 = Vertical_wall[f'wall{i+1}{j}'] , x = j , y = i , typew = 'v': creat_wall(arg1,arg2,x,y,typew))
62
63
```

در این قسمت تنظیم میشود که اگر هر کدام از دیوار های عمودی و افقی کلیک شد ، تابع Creat_wall فراخوانی شود.
این تابع ۵ ورودی میگیرد :
X : مختصات x دیوار کلیک شده
Y : مختصات y دیوار کلیک شده
Arg1 : دیوار کلیک شده به عنوان شیئ به تابع پاس داده میشود.
Arg2 : دیوار کناری به تابع پاس داده میشود ، که اگر دیوارهای عمودی کلیک شوند ، میشود دیوار زیرین و اگر دیوار های افقی کلیک شوند ، میشود دیوار سمت راستی.
Typew : که نوع دیوار کلیک شده را به تابع پاس میدهد. H برای دیوار افقی و v برای دیوار عمودی

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

```
64
65  class Maze :
66      X = 9
67      Y = 9
68      step = 0
69      number_of_p1_wall = 10
70      xplayer1 = 4
71      yplayer1 = 0
72      block = list()
73      for i in range(9) :
74          block.append([])
75          for j in range(9) :
76              block[i].append({
77                  'u' : False ,
78                  'r' : False ,
79                  'l' : False ,
80                  'd' : False ,
81                  'v' : -1
82              })
83      number_of_p2_wall = 10
84      xplayer2 = 4
85      yplayer2 = 8
```

X : بعد افقی بازی

y : بعد عمودی بازی

Step : گام بازی. این متغییر هر بار که یک بازیکن نوبت خود را پیش میرود یک عدد اضافه میشود

Number_of_p1_wall : تعداد اولیه دیوار بازیکن شماره ۱. هربار بازیکن یک دیوار بچیند ، این متغییر یک عدد کم میشود.

Xplayer1 , Yplayer1 : موقعیت اولیه بازیکن شماره ۱

Block : یک لیست از تمام خانه ها که اطلاعات هر خانه در این متغییر ذخیره میشود. مثل u دیوار بالا ، r دیوار راست ، l دیوار چپ ، d دیوار پایین و v ارزش هر خانه.

Number_of_p2_wall : تعداد اولیه دیوار بازیکن شماره ۲. هربار بازیکن یک دیوار بچیند ، این متغییر یک عدد کم میشود.

Xplayer2 , Yplayer2 : موقعیت اولیه بازیکن شماره ۲

آنالیز کد

ترسیم دیوارها ، خانه ها ، مهره ها و تعریف کلاس Maze
Line 1 To 101

```
86
87  player1 = tk.Button(Quoridor , bg = "#ff4d4d" ,borderwidth=3, relief="ridge")
88  player2 = tk.Button(Quoridor , bg = "#00ff80" ,borderwidth=3, relief="ridge" )
89  player1.place(height = 40 , width = 40 , x = 260 , y = 20)
90  player2.place(height = 40 , width = 40, x = 260 , y = 500 )
91  player1.config(command = lambda arg = player1 , p = 1 : set_yellow_squarGoal(arg , p))
92  player2.config(command = lambda arg = player2 , p = 2 : set_yellow_squarGoal(arg , p))
93
94  lbl_nwallp1 = tk.Label(Quoridor , bg = "white" , text = "Player1 Walls : 10" , font = ("Aria" ,9 , 'bold') , fg = 'red')
95  lbl_nwallp1.place(height = 30 , width = 120 , x = 120 , y = 570 , )
96
97  lbl_nwallp2 = tk.Label(Quoridor , bg = "white" , text = "Player2 Walls : 10" , font = ("Aria" ,9 , 'bold') , fg = "green")
98  lbl_nwallp2.place(height = 30 , width = 120 , x = 320 , y = 570)
99
100 lbl_turns = tk.Label(Quoridor , text = "Red Turn" , bg = "red"  ,font = ("Helvetica" , 8 , "bold") ,borderwidth=2, relief="solid")
101 lbl_turns.place(height = 30 , width = 80 , x = 240 , y = 570)
102
```

در این قسمت مهره ی بازیکن ها و چند label تعریف ترسیم میشوند. با کلیک بر روی باتن مهره ها ، تابع Set_yellowGoal() فراخوانی میشود. این تابع مهره را به عنوان شیئ و هم چنین یک عدد به نام p (اگر مهره اول باشد p=1 اگر مهره دوم p=2) به عنوان ورودی میگیرد.

Lbl_nwalp1 : یک لیبل که تعداد دیوار های باقی مانده بازیکن شماره یک را میگوید.
Lbl_nwallp2 : یک لیبل که تعداد دیوار های باقی مانده بازیکن شماره دو را میگوید.
Lbl_turns : یک لیبل که اعلام میکند نوبت کدام بازیکن است.

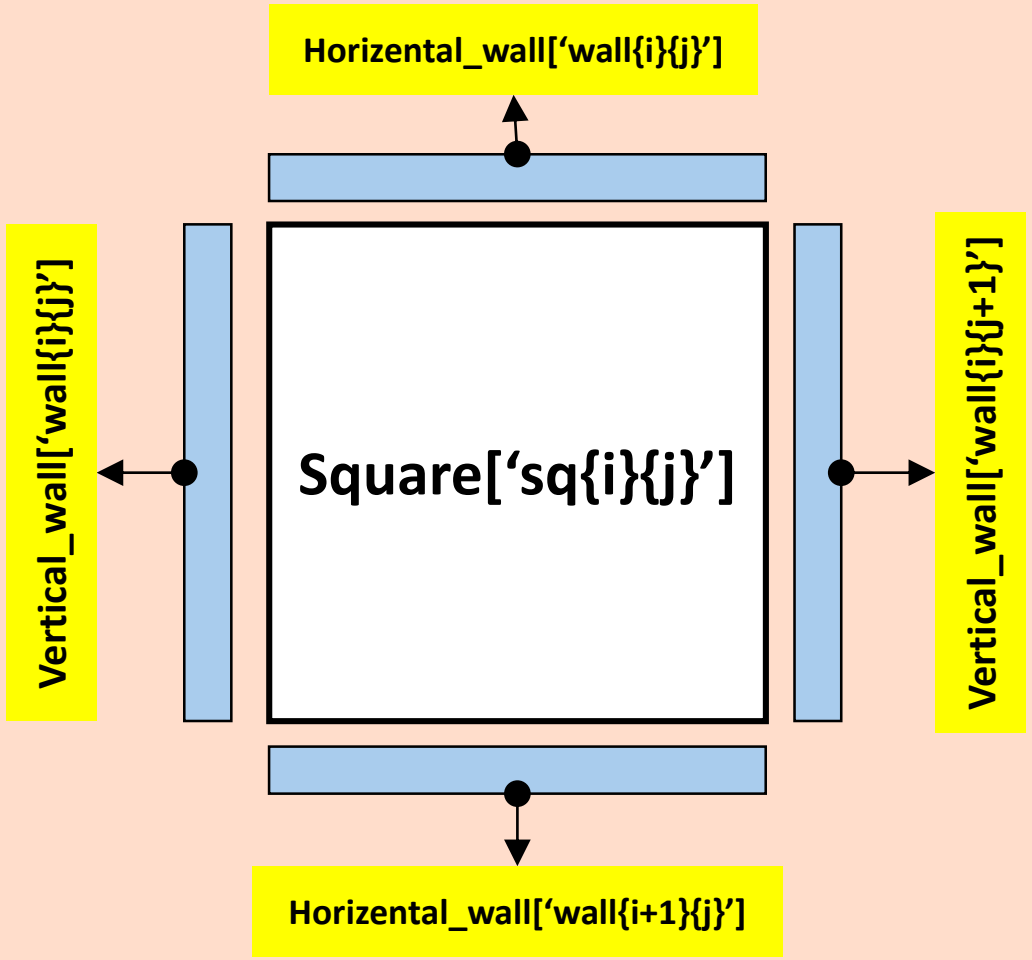
آنالیز کد

assign_wall_to_block()
Line 103 To 121

در این تابع ، با هر بار فراخوانی در تمام
خانه ها جست و جو کرده و اطلاعات دیوار
هر خانه را در متغییر بلاک ذخیره میکنم.

```
102
103  def assign_wall_to_block() :
104      for i in range(9) :
105          for j in range(9) :
106              if Horizental_wall[f'wall{i}{j}']['bg'] != '#aee2ff'  : #light blue
107                  Maze.block[i][j]['u'] = True
108              else :
109                  Maze.block[i][j]['u'] = False
110              if Horizental_wall[f'wall{i+1}{j}']['bg'] != '#aee2ff' :
111                  Maze.block[i][j]['d'] = True
112              else :
113                  Maze.block[i][j]['d'] = False
114              if Vertical_wall[f'wall{i}{j}']['bg'] != '#aee2ff' :
115                  Maze.block[i][j]['l'] = True
116              else :
117                  Maze.block[i][j]['l'] = False
118              if Vertical_wall[f'wall{i}{j+1}']['bg'] != '#aee2ff' :
119                  Maze.block[i][j]['r'] = True
120              else :
121                  Maze.block[i][j]['r'] = False
122      #for i in range(9):
123          #for j in range(9):
```

اگر رنگ باتن ها ابی کم رنگ بود ، یعنی
دیوار نیست (False) در غیر این صورت
دیوار هست (True)



آنالیز کد

clear_square_color() : 127 To 130
Reset_vblock() : 131 To 134

```
126
127 def clear_square_color() :
128     for i in range(9) :
129         for j in range(9) :
130             Square[f'sq{i}{j}']['bg'] = "white"
131 def reset_vblock() :
132     for i in range(9) :
133         for j in range(9) :
134             Maze.block[i][j]['v'] = -1
```

در تابع clear_square_color() رنگ تمام خانه های بازی به رنگ اولیه خود یعنی سفید بازمیگردند. زمانی که روی مهره حرکت میکنیم و خانه های مجاز برای حرکت مهره به رنگ زرد در میاند ، حالا اگر مهره حرکت کند باید رنگ های زرد به رنگ سفید تغییر یابند.

از تابع reset_vblock() برای منفی یک کردن ارزش همه ی خانه ها استفاده میشود. زمانی که الگوریتم محاصره شدن اجرا شود ، ارزش خانه دچار تغییر میشود و اگر دوباره بخواهیم از این الگوریتم استفاده کنیم ابتدا باید این تابع را فراخوانی کنیم تا ارزش همه ی خانه ها منفی یک شود سپس الگوریتم اجرا شود.

آنالیز کد

is_valid_wall(*x , y , typew*)
Line 135 To 240

```
135 def is_valid_wall(x , y , typew) :
136
137     #return true or false
138     assign_wall_to_block()
139     if typew == 'h' :
140         Maze.block[y][x]['u'] = True
141         Maze.block[y][x+1]['u'] = True
142         Maze.block[y-1][x]['d'] = True
143         Maze.block[y-1][x+1]['d'] = True
144     elif typew == 'v' :
145         Maze.block[y][x]['l'] = True
146         Maze.block[y+1][x]['l'] = True
147         Maze.block[y][x-1]['r'] = True
148         Maze.block[y+1][x-1]['r'] = True
149
150     else :
151         messagebox.showinfo("error - is_valid_wall()" , "typew is not h or v")
152
153     ############################################ p1
154     reset_vblock()
155     validp1 = False
156     Maze.block[Maze.yplayer1][Maze.xplayer1]['v'] = 0
157     step = 0
158     while True :
159         f = False
160         for i in range(9) :
161             for j in range(9) :
162
163                 if Maze.block[i][j]['v'] == step :
164                         #messagebox.showinfo("v" , f'{i} {j} : {step}')
165                         if Maze.block[i][j]['u'] == False:
166                             if Maze.block[i-1][j]['v'] == -1 :
167                                 #messagebox.showinfo("v" , f'u {i-1} {j} : {step + 1}')
168                                 Maze.block[i-1][j]['v'] = step + 1
169                                 f = True
170                         if Maze.block[i][j]['r'] == False  :
171                             if Maze.block[i][j+1]['v'] == -1 :
172                                 #messagebox.showinfo("v" , f'r {i} {j+1} : {step + 1}')
173                                 Maze.block[i][j+1]['v'] = step + 1
174                                 f = True
175                         if Maze.block[i][j]['l'] == False :
176                             if Maze.block[i][j-1]['v'] == -1 :
177                                 #messagebox.showinfo("v" , f'l {i} {j-1} : {step + 1}')
178                                 Maze.block[i][j-1]['v'] = step + 1
179                                 f = True
180                         if Maze.block[i][j]['d'] == False :
181                             if Maze.block[i+1][j]['v'] == -1 :
182                                 #messagebox.showinfo("v" , f' d {i+1} {j} : {step + 1}')
183                                 Maze.block[i+1][j]['v'] = step + 1
184                                 f = True
185         step += 1
186         if f == False :
187             break
188     for i in range(9) :
189         #messagebox.showinfo(f'v block8{i}' , Maze.block[8][j]['v'])
190         if Maze.block[8][i]['v'] != -1 :
191
192             validp1 = True
193             break
```

این تابع الگوریتم محاصره شدن را اجرا میکند. این تابع مختصات دیواری که قرار است ترسیم شود هم چنین تایپ ان (عمودی یا افقی) را به عنوان ورودی میگیرد ، سپس در متغییر block فرض میکند که این دیوارد ترسیم شده ، سپس الگوریتم را اجرا میکند. اگر ربات یک محاصره شود ارزش validp1= false میشود.

برای مهره دوم هم همین روند پیش میرود. اگر مهره 2 محاصره شود ارزش validp2 = false میشود.

در نهایت اگر ارزش validp1 و validp2 هر دو Trueبود (یعنی با این دیوار هیچ کدام از دو مهره محاصره نمیشوند) تابع مقدار True برمیگرداند که یعنی ترسیم این دیوار مشکلی ندارد.

عکس تنها مربوط به الگوریتم محاصره شدن برای بازیکن 1 است

آنالیز کد

creat_wall(*arg1 , arg2 , x , y , typew*)
Line 242 To 320

```
241
242  def creat_wall(arg1 , arg2 , x , y , typew) :
243      #messagebox.showinfo(" x , y" , f'{x}  {y}')
244      if Maze.yplayer2 == 0 :
245          messagebox.showinfo("End Game" , "Player2 Won")
246      elif Maze.yplayer1 == 8 :
247          messagebox.showinfo("End Game" , "Player1 Won")
248      else :
249          if Maze.step % 2 == 0 :
250              if arg1['bg'] == "#aee2ff" and arg2['bg'] == "#aee2ff" and Maze.number_of_p1_wall != 0 :
251                  clear_square_color()
252                  if is_valid_wall(x,y,typew) :
253
254                      if typew == 'h' :
255                          if Vertical_wall[f'id{y-1}{x+1}'] != Vertical_wall[f'id{y}{x+1}'] or Vertical_wall[f'id{y-1}{x+1}'] == -1 :
256                              arg1['bg'] = "#e8002a" #red
257                              arg2['bg'] = "#e8002a"
258                              #Horizental_wall[f'wall{y}{x}'].lift()
259                              #Horizental_wall[f'wall{y}{x+1}'].lift()
260                              Horizental_wall[f'wall{y}{x}'].place(height = 10 , width = 60 , x = (x) * 50 +(x+1)*10 + 5, y = (y)*50 + (y) * 10 +5)
261                              Horizental_wall[f'id{y}{x}'] = Maze.step
262                              Horizental_wall[f'id{y}{x+1}'] = Maze.step
263                              Maze.step += 1
264                              Maze.number_of_p1_wall -= 1
265                              lbl_nwallp1['text'] = f'Player1 Walls : {Maze.number_of_p1_wall}'
266                              lbl_turns['bg'] = "#00d021"
267                              lbl_turns['text'] = "Green Turn"
268                      else :
269                          if Horizental_wall[f'id{y+1}{x-1}'] != Horizental_wall[f'id{y+1}{x}'] or Horizental_wall[f'id{y+1}{x-1}'] == -1 :
270                              arg1['bg'] = "#e8002a" #red
271                              arg2['bg'] = "#e8002a"
272                              #Vertical_wall[f'wall{y}{x}'].lift()
273                              #Vertical_wall[f'wall{y+1}{x}'].lift()
274                              Vertical_wall[f'wall{y}{x}'].place(height = 60 , width = 10 , x = x * 50 + (x)*10 + 5, y = (y)*50+(y+1)*10 + 5)
275                              Vertical_wall[f'id{y}{x}'] = Maze.step
276                              Vertical_wall[f'id{y+1}{x}'] = Maze.step
277
278                              Maze.step += 1
279                              Maze.number_of_p1_wall -= 1
280                              lbl_nwallp1['text'] = f'Player1 Walls : {Maze.number_of_p1_wall}'
281                              lbl_turns['bg'] = "#00d021"
282                              lbl_turns['text'] = "Green Turn"
283                      assign_wall_to_block()
284      else :
```

در این تابع ابتدا بررسی میشود که باقی مانده ی maze.step بر 2 چند میشود.
اگر صفر شد نوبت بازیکن شماره 1 است و رنگ دیوار به رنگ قرمز در میاید و اگر باقی مانده 1 شد یعنی نوبت باریکن شماره 2 است و رنگ دیوار به رنگ سبز در خواهد امد. سپس چک میکند که دیواری که انتخاب شده و دیوار کناری ان هر دو به رنگ ابی کم رنگ باشد تا بشود دیوار گذاشت هم چنین چک میکند تعداد دیوارهای باقی مانده ی هر بازیکن صفر نباشد تا مجاز باشد برای چیدن دیوار. سپس تابع is_valis_wall() فراخوانی میشود تا درستی دیواری که قرار است ترسیم شود چک شود.
اگر همه ی شروط بالا مجاز بودند ، شرط روی هم نبودن دیوار ها و یکسان نبودن ارزش دیوار های کناری (که در قسمت الگوریتم دیوار گذاشتن توضیح داده شد) چک میشود. اگر این شرط هم درست بود : به ترسیم کردن دیوار میپردازد.
از تعداد دیوارهای باقی مانده هر بازیکن یک عدد کم میشود.
متغییر maze.step یک عدد افزایش میابد.
و ارزش دو باتنی که در مجموع یک دیوار را تشکیل داده اند برابر با maze.step قرار میگیرد.

آنالیز کد

set_yellow_squarGoal(*arg,p*)
Line 321 To 410

```
321 def set_yellow_squarGoal(arg,p) :
322     #messagebox.showinfo("1" ,f'{Maze.yplayer1} {Maze.xplayer1}')
323     #messagebox.showinfo("2" , f'{Maze.yplayer2} {Maze.xplayer2}')
324     if Maze.yplayer1 == 8 :
325         messagebox.showinfo("End Game" , "Player1 Won")
326     elif Maze.yplayer2 == 0 :
327         messagebox.showinfo("End Game" , "Player2 Won")
328     else :
329         if Maze.step % 2 == 0 :
330             if p == 1 :
331                 #messagebox.showinfo("p1 xy" , f'{Maze.xplayer1}  {Maze.yplayer1}')
332                 clear_square_color()
333                 if Horizental_wall[f'wall{Maze.yplayer1}{Maze.xplayer1}']['bg'] == "#aee2ff" : #up
334                     #messagebox.showinfo("up")
335                     #messagebox.showinfo("p1 xy" , f'{Maze.xplayer1}   {Maze.yplayer1}')
336                     if Maze.yplayer1 - 1 == Maze.yplayer2 and Maze.xplayer1 == Maze.xplayer2 :
337                         if Horizental_wall[f'wall{Maze.yplayer1-1}{Maze.xplayer1}']['bg'] == "#aee2ff" :
338                             Square[f'sq{Maze.yplayer1-2}{Maze.xplayer1}']['bg'] = "#ffff79"
339                     else :
340                         Square[f'sq{Maze.yplayer1-1}{Maze.xplayer1}']['bg'] = "#ffff79"
341
342                 if Horizental_wall[f'wall{Maze.yplayer1+1}{Maze.xplayer1}']['bg'] == "#aee2ff" : #down
343                     #messagebox.showinfo("down")
344                     #messagebox.showinfo("p1 xy" , f'{Maze.xplayer1}   {Maze.yplayer1}')
345                     if Maze.yplayer1+1 == Maze.yplayer2 and Maze.xplayer1 == Maze.xplayer2 :
346                         if Horizental_wall[f'wall{Maze.yplayer1+2}{Maze.xplayer1}']['bg'] == "#aee2ff" :
347                             Square[f'sq{Maze.yplayer1+2}{Maze.xplayer1}']['bg'] = "#ffff79"
348                     else :
349                         Square[f'sq{Maze.yplayer1+1}{Maze.xplayer1}']['bg'] = "#ffff79"
350
351                 if Vertical_wall[f'wall{Maze.yplayer1}{Maze.xplayer1}']['bg'] == "#aee2ff" : #left
352                     #messagebox.showinfo("left")
353                     #messagebox.showinfo("p1 xy" , f'{Maze.xplayer1}   {Maze.yplayer1}')
354                     if Maze.xplayer1 - 1 == Maze.xplayer2 and Maze.yplayer1 == Maze.yplayer2 :
355                         if Vertical_wall[f'wall{Maze.yplayer1}{Maze.xplayer1 -1}']['bg'] == "#aee2ff" :
356                             Square[f'sq{Maze.yplayer1}{Maze.xplayer1-2}']['bg'] = "#ffff79"
357                     else :
358                         Square[f'sq{Maze.yplayer1}{Maze.xplayer1-1}']['bg'] = "#ffff79"
359
360                 if Vertical_wall[f'wall{Maze.yplayer1}{Maze.xplayer1+1}']['bg'] == "#aee2ff" : #right
361                     #messagebox.showinfo("right")
362                     #messagebox.showinfo("p1 xy" , f'{Maze.xplayer1}   {Maze.yplayer1}')
363                     if Maze.xplayer1 + 1 == Maze.xplayer2 and Maze.yplayer1 == Maze.yplayer2 :
364                         if Vertical_wall[f'wall{Maze.yplayer1}{Maze.xplayer1+2}']['bg'] == "#aee2ff" :
365                             Square[f'sq{Maze.yplayer1}{Maze.xplayer1+2}']['bg'] = "#ffff79"
366                     else :
```

این تابع وظیفه ی زرد کردن خانه های اطراف هر مهره را برای حرکت دارد.

برای مثال اگر maze.step %2 = 0 یعنی نوبت بازیکن یک بود هم چنین p = 1 یعنی روی باتن مهره یک کلیک شده بود. همه ی خانه های اطراف مهره یک در صورتی که دیوار نبود زرد رنگ میشوند.

اگر در خانه ی مجاور مهره ی حریف بود ، خانه ی بقلی ان زرد رنگ میشود.

آنالیز کد

move(*arg* , *y* , *x*)
Line 413 To 471

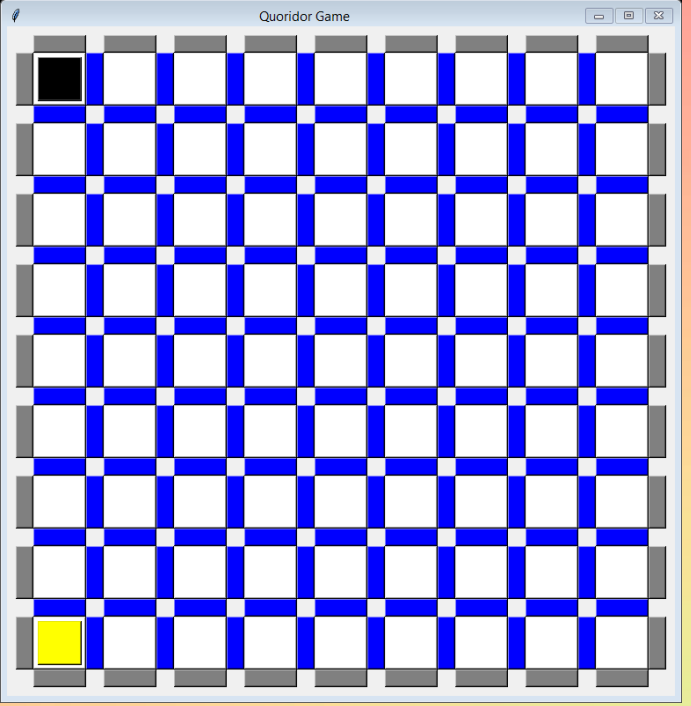
```
412
413  def move(arg , y , x) :
414      #messagebox.showinfo("block xy " , f'{x}  {y}')
415      gy = 0
416      gx = 0
417      xx = 0
418      yy = 0
419      if arg['bg'] == "#ffff79" :
420
421          if Maze.step % 2 == 0 :
422
423
424
425              xx = x - Maze.xplayer1
426              gx = Maze.xplayer1 + xx
427              Maze.xplayer1 += xx
428
429              yy = y - Maze.yplayer1
430              gy = Maze.yplayer1 + yy
431              Maze.yplayer1 += yy
432
433              player1.place(x = (gx+1)*10 + gx*50 + 10 , y = (gy+1)*10 + gy*50 + 10)
434              clear_square_color()
435              Maze.step += 1
436              lbl_turns['bg'] = "#00d021"
437              lbl_turns['text'] = "Green Turn"
438              if Maze.yplayer1 == 8 :
439
440                  lbl_turns['bg'] = "#b4a8ff"
441                  lbl_turns['text'] = "Player1 Won"
442                  for i in range(9) :
443                      Square[f'sq{8}{i}']['bg'] = "#ffa8a8"
444
445
446                  messagebox.showinfo("End Game" , "Player1 Won")
447
448          ########################################################## p2
449          else :
450
451              xx = x - Maze.xplayer2
452              gx = Maze.xplayer2 + xx
453              Maze.xplayer2 += xx
454
455
456
457              yy = y - Maze.yplayer2
```

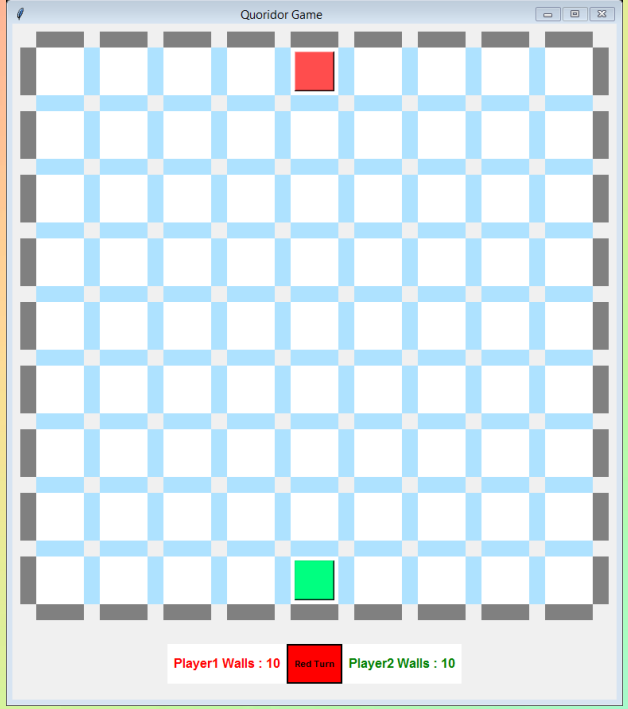
این تابع زمانی که روی خانه ها کلیک میشود اجرا میشود. اگر رنگ خانه ی کلیک شده زرد رنگ بود و برای مثال نوبت بازیکن یک بود یعنی باید بازیکن شماره یک در خانه ی زرد رنگ کلیک شده قرار بگیرد. این تابع مختصات خانه ی کلیک شده را به عنوان ورودی میگیرد. از تفاضل مختصات مهره بازیکن و خانه ، حرکت مشخص میشود و سپس با متد place ، مهره را در مختصات جدید قرار میدهیم.

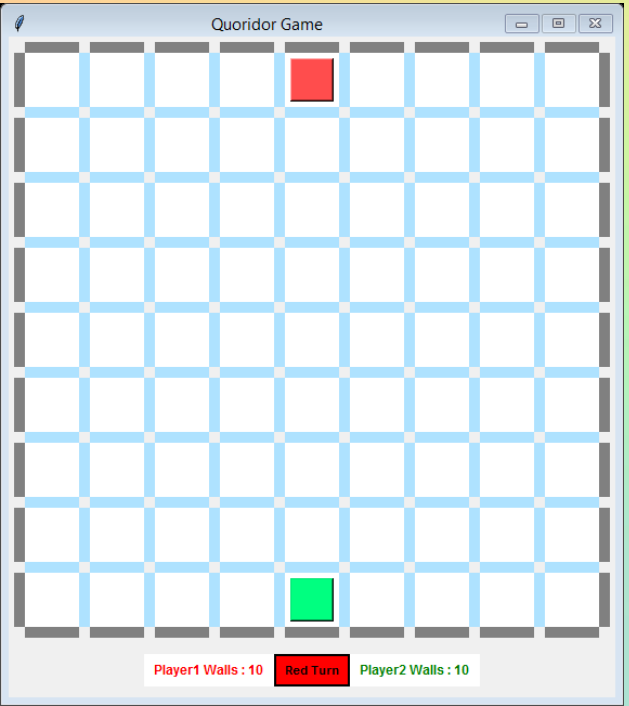
اگر مهره بعد از حرکت در انتهای خانه های رو به رویی قرار گرفت ، بازی تمام میشود.

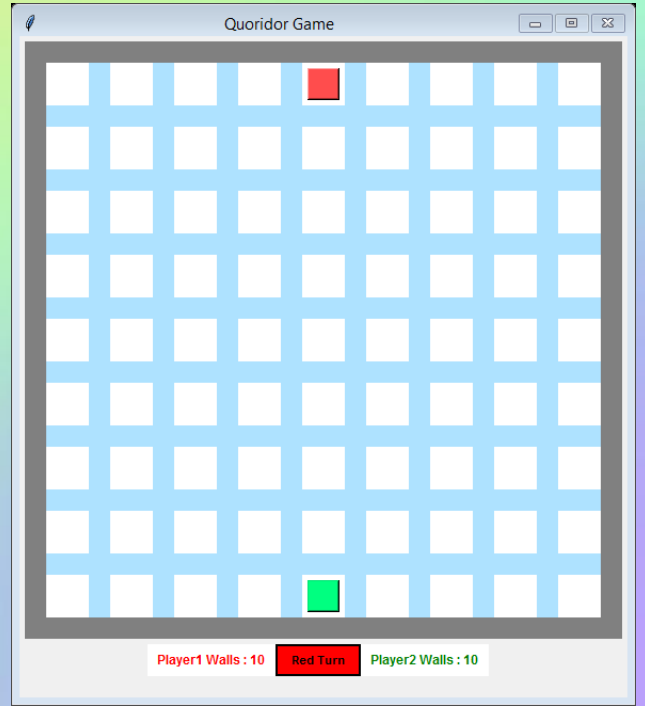
برای مثال اگر مختصات y بازیکن شماره یک بعد از حرکت شد 8 ، پیغام اتمام بازی فعال و رنگ خانه های ردیف 8 به رنگ قرمز در میاید.

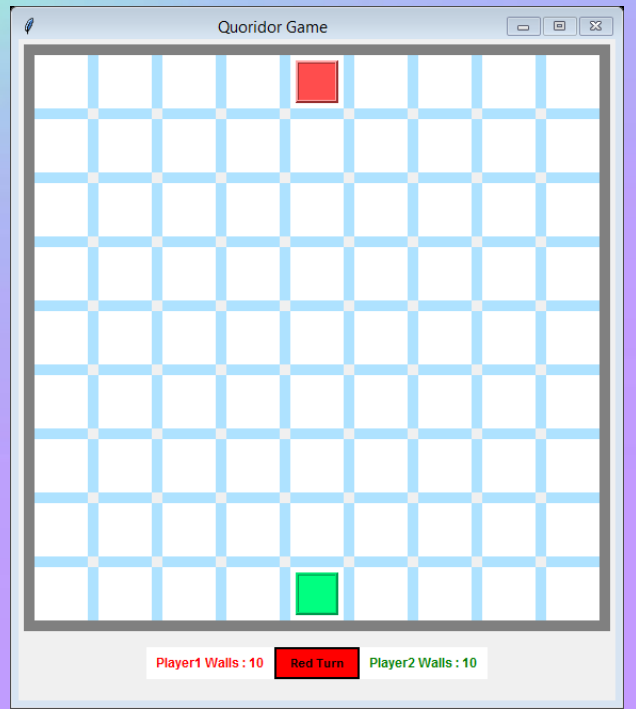
نسخه های مختلف بازی

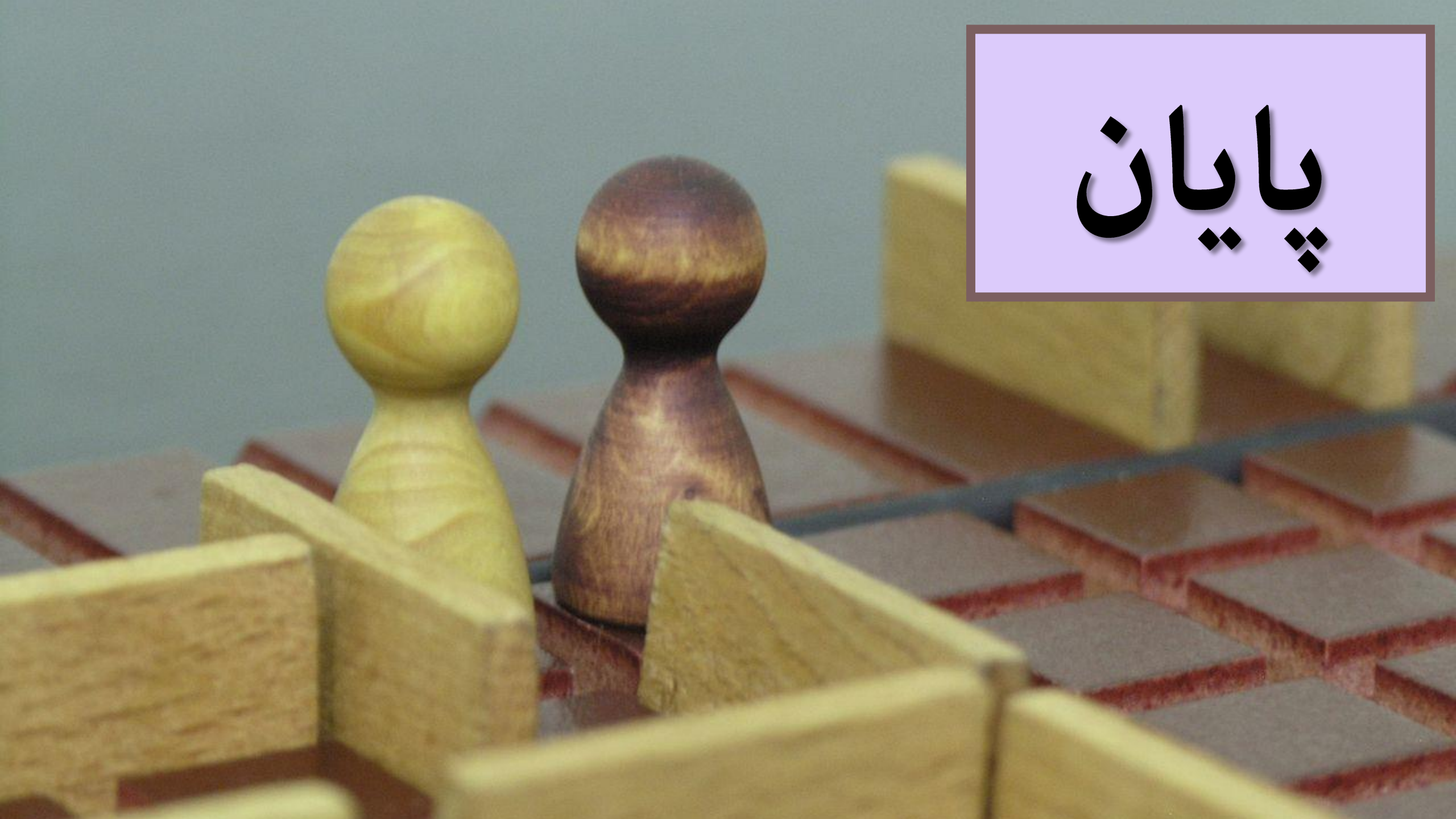











پایان