پیاده سازی این پروژه به اینصورت است که باید یک تابع minimax و یک تابع heuristic تعریف کنیم بصورتیکه بازیکن قرمز در برابر دشمن خود که بازی رندوم دارد، برنده شود.

تابع minimax یک تابع بازگشتی است، به همین دلیل در ابتدای آن شرط صفر شدن عمق و تعداد حرکت هارا چک میکنیم. وقتی که نوبت بازیکن قرمز است تابع ما باید مقدار ماکسیمم را پیدا کند و وقتی که نوبت بازیکن آبی است، مقدار مینیمم را پیدا کند. قبل از وارد شدن به تابع minimax باید مسیر انتخاب شده را از available_moves حذف کرده و به نقاط blue/red اضافه کنیم. سپس در مسیر بازگشت این تابع بازگشتی دوباره مسیر را به available_moves اضافه کرد و از لیست red/blue حذف میکنیم. سپس با توجه به مینیمم و ماکسیمم بودن مقدار دریافتی، شروط را بررسی کرده و مقادیر متغییرهارا تعیین میکنیم.

```
if player_turn == 'red':
    value = -math.inf
    bestMove = None
    for move in (self.available_moves):
        self.turn = self._swap_turn(self.turn)
        self.available_moves.remove(move)
        self.red.append(move)
        unused, tempValue =  self.minimax(depth - 1, self.turn)
        self.red.remove(move)
        self.available_moves.append(move)
        self.turn = self._swap_turn(self.turn)
        if tempValue >= value:
            value = tempValue
            bestMove = move
        alpha = max(alpha, value)
        if self.prune and beta <= alpha:
            break
    return bestMove, value
```

```
  elif player_turn == 'blue':
      value = math.inf
      for move in (self.available_moves):
          self.turn = self._swap_turn(self.turn)
          self.available_moves.remove(move)
          self.blue.append(move)
          unused, tempValue =  self.minimax(depth - 1, self.turn)
          self.blue.remove(move)
          self.available_moves.append(move)
          self.turn = self._swap_turn(self.turn)
          if tempValue <= value:
              value = tempValue
              bestMove = move
          beta = min(beta, value)
          if self.prune and beta >= alpha:
              break
      return bestMove, value
```

تابع heuristic به صورتی تعریف میشود که در رقابت بازیکن قرمز و آبی، اگر نوبت بازیکن قرمز باشد دو حالت ممکن است پیش بیاید: یک اینکه در حرکت بعدی آبی برنده شود، که در اینصورت چون قرمز باخته است، ۸- امتیاز به مجموع امتیازات اضافه شده و دو اینکه نه حرکت بعدی باعث برد آبی نشود که دراینصورت ۱۰+ امتیاز به مجموع امتیازات اضافه میشود. به همین صورت اگر نوبت بازیکن آبی باشد نیز دو حالت ممکن است پیش بیاید: یک اینکه در حرکت بعدی قرمز برنده شود، که در اینصورت ۱۰+ امتیاز به مجموع امتیازات اضافه میشود و دو اینکه در حرکت بعدی قرمز برنده نشود، که در اینصورت چون حرکتی از سمت قرمز انجام شده که منجر به برد او نشده ۸- امتیاز به مجموع امتیازات اضافه میشود.

```
elif self.turn == 'red':
    for move in (self.available_moves):
        self.available_moves.remove(move)
        self.red.append(move)
        if self.gameover(self.red, self.blue) == 'blue':
            count -= 8
        elif self.gameover(self.red, self.blue) == 0:
            count += 10
        self.available_moves.append(move)
        self.red.remove(move)
    return count
```

```
elif self.turn == 'blue':
    for move in (self.available_moves):
        self.available_moves.remove(move)
        self.blue.append(move)
        if self.gameover(self.red, self.blue) == 'red':
            count += 10
        elif self.gameover(self.red, self.blue) == 0:
            count -= 8
        self.available_moves.append(move)
        self.blue.remove(move)
    return count
```

هرس آلفا و بتا در ابتدای تابع minimax مقدار آلفا را برابر ∞- و مقدار بتا را برابر ∞+ میگذاریم. زمانی که نوبت بازیکن قرمز باشد، مقدار آلفا برابر مقدار ماکسیمم آلفا و value میشود و زمانی که نوبت بازیکن آبی باشد، مقدار بتا برابر مقدار مینیمم مقدار بتا و value میشود. سپس شرط True بودن Prune و beta <= alpha برای نوبت قرمز و beta >= alpha برای نوبت آبی را بررسی میکنیم.

Output without Prune

Depth = 1

```
PS D:\Artificial intelligence\Project3> python main.py 1 0
D:\Artificial intelligence\Project3\main.py:3: DeprecationWarning:
thon
  from symbol import dotted_as_name
{'red': 93, 'blue': 7}
Probability of Winning: 0.93
Time of each Play: 0.0013605022430419921
Total time : 0.13605022430419922
```

Depth = 3

```
PS D:\Artificial intelligence\Project3> python main.py 3 0
D:\Artificial intelligence\Project3\main.py:3: DeprecationWarning:
thon
  from symbol import dotted_as_name
{'red': 80, 'blue': 20}
Probability of Winning: 0.8
Time of each Play: 0.12489874124526977
Total time : 12.490875482559204
```

Depth = 5

```
PS D:\Artificial intelligence\Project3> python main.py 5 0
D:\Artificial intelligence\Project3\main.py:3: DeprecationWarning:
thon
  from symbol import dotted_as_name
{'red': 76, 'blue': 24}
Probability of Winning: 0.76
time : 2070.6941361427307
```

Output with Prune

Depth = 1

```
PS D:\Artificial intelligence\Project3> python main.py 1 0
D:\Artificial intelligence\Project3\main.py:3: DeprecationWarning:
thon
  from symbol import dotted_as_name
{'red': 94, 'blue': 6}
Probability of Winning: 0.94
Time of each Play: 0.0012924480438232422
Total time : 0.12924480438232422
```

Depth = 3

```
PS D:\Artificial intelligence\Project3> python main.py 3 0
D:\Artificial intelligence\Project3\main.py:3: DeprecationWarning:
thon
  from symbol import dotted_as_name
{'red': 90, 'blue': 10}
Probability of Winning: 0.9
Time of each Play: 0.009675135612487793
Total time : 0.9675135612487793
```

Depth = 5

```
PS D:\Artificial intelligence\Project3> python main.py 5 0
D:\Artificial intelligence\Project3\main.py:3: DeprecationWarning:
thon
  from symbol import dotted_as_name
{'red': 84, 'blue': 16}
Probability of Winning: 0.84
Time of each Play: 0.040528335571289 06
Total time : 4.052833557128906
```

Depth = 7

```
PS D:\Artificial intelligence\Project3> python main.py 7 0
D:\Artificial intelligence\Project3\main.py:3: DeprecationWarning:
thon
  from symbol import dotted_as_name
{'red': 80, 'blue': 20}
Probability of Winning: 0.8
Time of each Play: 0.012689843177795 41
Total time : 1.268984317779541
```

بخش سوالات

۱. یک heuristic خوب چه ویژگیهایی دارد؟ علت انتخاب heuristic شما و دلیل برتری آن نسبت به تعدادی از روشهای دیگر را بیان کنید.

ویژگی یک هیوریستیک خوب این است که بتواند پیشبینی خوبی نسبت به حرکات آینده داشته باشد. در اینجا ما نیز همین کار را کردیم به اینصورت که هر بار مسیرهای موجود را بررسی میکنیم تا ببینیم در هر مسیر چه اتفاقی می افتد. در این هیوریستیک به اینصورت عمل میکنیم که هر حرکت از حرکت هایی که برای بازیکن فعلی موجود است را انتخاب کرده و برای هرکدام بررسی میکنیم که با انجام این حرکت مثلثی تشکیل میشود یا خیر. در صورت تشکیل شدن یا نشدن مثلث امتیازی به مجموع امتیازات اضافه یا کم میشود که در بالا توضیح داده شده است. علت انتخاب این هیوریستیک این است که طبق درصد گرفته شده در اکثر مواقع میتواند باعث برنده شدن بازیکن قرمز شود که میتواند مارا به هدفمان برساند.

۲. آیا میان عمق الگوریتم و پارامترهای حساب شده روابطی میبینید؟ به طور کامل بررسی کنید که عمق الگوریتم چه تاثیراتی بر روی شانس پیروزی، زمان و گره های دیده شده میگذارد.

در این بازی بازیکن ما با فرض اینکه بازیکن حریف به صورت بهینه بازی میکند، پیش میرود. بنابراین وقتی عمق های بیشتری را میبینند، با فرض اینکه حرکتی که حریف میزند، بهینه است، حرکت خودرا پیش میبرد و اگر در یک مرحله بازیکن باخت خودش را ببیند، طوری عمل میکند که ببازد تا امتیاز کمتری از دست بدهد اما در واقعیت چون حرکت رقیب رندوم است و بهینه عمل نمیکند، ممکن بود حرکتی را بازی کند که باعث باخت خودش شود و بازیکن ما برنده شود. به همین دلیل هرچه بازیکن ما عمق بیشتری را بررسی کند احتمال بردش کاهش میابد. تاثیری که عمق روی زمان بازی میگذارد به اینصورت است که هرچه تعداد عمق افزایش یابد، زمان بازی نیز طبیعتا بیشتر خواهد شد، زیرا تعداد گره هایی که باید بررسی شوند، افزایش میابد.

۳. **وقتی از روش هرس کردن استفاده میکنید، برای هر گره درخت، فرزندانش به چه ترتیبی اضافه میشوند؟ آیا این ترتیب اهمیت دارد؟ چرا این ترتیب را انتخاب کردید؟**

هرس کردن زمانی به ما کمک میکند که ترتیب چینش بصورتی باشد که گره های قابل حذف در ابتدای چینش قرار گرفته باشند، در اینصورت باقی گره ها حذف شده و نیاز به چک کردن آنها نخواهد بود. که در این حالت زمان اجرا کاهش میابد. در مسئله ما نیز چون به ترتیب موقعیت راس ها، حرکت ها انجام میشوند و امتیاز کسب شده در مسیر هایی که باعث ایجاد مثلث نشوند، یکسان است، بنابراین نیازی به چک کردن تمامی مسیرهای موجود نخواهد بود و میتوان از یک جایی به بعد درخت را هرس کرد. در نتیجه زمان اجرا کاهش میابد.