Escape BOOM!

by Sepehr Abbaspour - 610398147

لازمه یک برنامه درست، حرفه ای و یا جذاب، همواره یک محوریت پیچیده علمی و شامل هزاران خط، توابع، ماژول های مختلف و ناشناخته نیست.

گاهی یک ایده یا موضوع نسبتاً ساده می تواند چنان چالش برانگیز باشد که با وجود تعداد خطوط کمتر وعدم وجود توابع و ماژول های متعدد، برنامه ای قدرتمند را برایمان به ارمغان آورد.

موضوع این پروژه به ظاهر ساده، چالش هایی را در عمل به وجود آورد که به واسطه علم و پشتکار بنده، پشت سر گذاشته شد؛ موضوعی که در ادامه با تمامی جزئیات به آن خواهیم پرداخت.

بازی ای را تصور کنید که شامل دو بازیکن با سلامتی کامل (100 واحد) است.

هر بازیکن در نوبت خود می تواند یکی ز سه دری که در مقابل او قرار دارد، یک در را اختیار کند به طوریکه:

در اول شامل یک مسئله ریاضی و محاسباتی است.

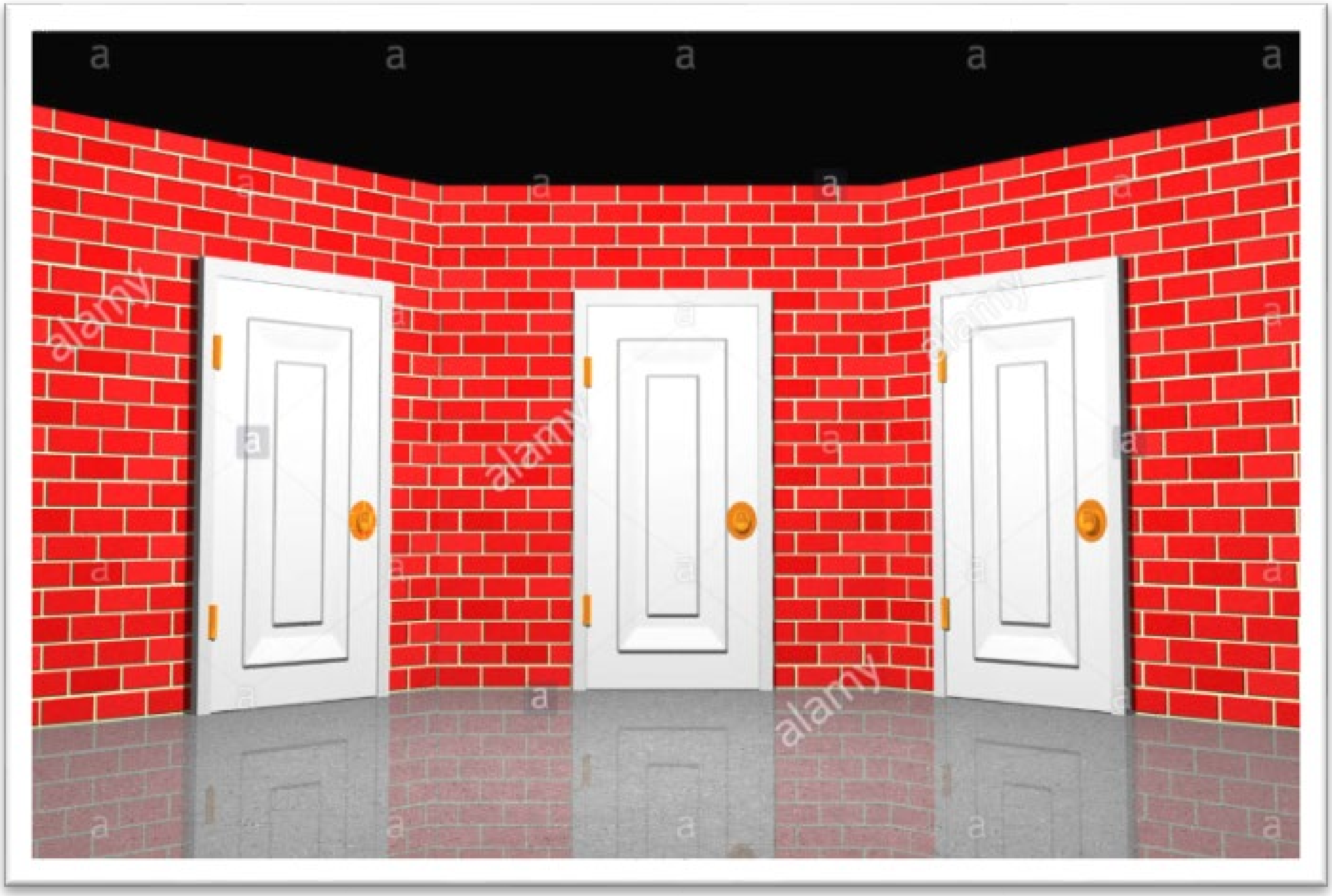
در دوم شامل یک معما است.

در سوم شامل چهار بازی است که به انتخاب بازیکن، یکی از بازی ها فعال شده و حریف، ملزوم به شرکت در بازی منتخب است.

هر بار بازیکنی که دری را انتخاب کرده است، موفقیت بدست آورد، از سلامتی حریف 10 واحد کمتر می شود. در غیر آن صورت، از سلامتی او 10 واحد کسر می شود.

اگر بازیکنی در سوم را انتخاب کرده و موفقیت به دست نیاورد، به احترام حریف او که بی اختیار در بازی شرکت کرده است، از سلامتی آن بازیکن 20 وحد کم خواهد شد.

لازم به ذکر است بازیکن شماره 1 و بازیکن شماره 2 در هر بازی، به ترتیب شروع کننده بازی و حریف او است.



حال به بررسی الگوریتم های تشکیل دهنده این بازی می پردازیم:

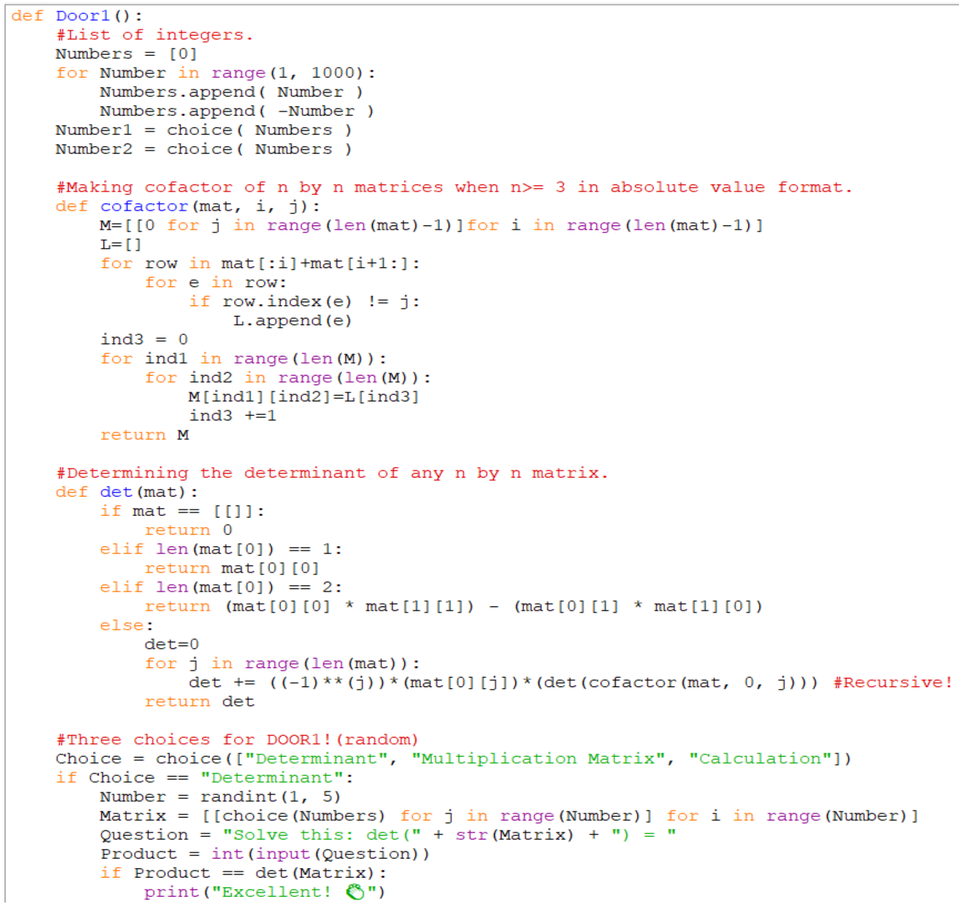
به منظور استفاده از هوش مصنوعی در سطح پایین، یکبار برای همیشه ماژول random و محتویات آن را تعریف می کنیم:

```
from random import *
```

توابعی را به عنوان معرف هر در ساخته و قید می کنیم اگر ارزش هر یک از توابع به ترتیب "Starter WON!" و "Starter LOST!" بود، بازیکنی که آن در را انتخاب کرده است به ترتیب پیروز و بازنده در نوبت خود خواهد بود.

چگونگی رسیدن به هر یک از این مقادیر در تعریف توابع آمده است که به صورت زیر بررسی می کنیم.

تابع مختص در اول



```python
def Door1():
    #List of integers.
    Numbers = [0]
    for Number in range(1, 1000):
        Numbers.append( Number )
        Numbers.append( -Number )
    Number1 = choice( Numbers )
    Number2 = choice( Numbers )

    #Making cofactor of n by n matrices when n>= 3 in absolute value format.
    def cofactor(mat, i, j):
        M=[[0 for j in range(len(mat)-1)]for i in range(len(mat)-1)]
        L=[]
        for row in mat[:i]+mat[i+1:]:
            for e in row:
                if row.index(e) != j:
                    L.append(e)
        ind3 = 0
        for ind1 in range(len(M)):
            for ind2 in range(len(M)):
                M[ind1][ind2]=L[ind3]
                ind3 +=1
        return M

    #Determining the determinant of any n by n matrix.
    def det(mat):
        if mat == [[]]:
            return 0
        elif len(mat[0]) == 1:
            return mat[0][0]
        elif len(mat[0]) == 2:
            return (mat[0][0] * mat[1][1]) - (mat[0][1] * mat[1][0])
        else:
            det=0
            for j in range(len(mat)):
                det += ((-1)**(j))*(mat[0][j])*(det(cofactor(mat, 0, j))) #Recursive!
            return det

    #Three choices for DOOR1!(random)
    Choice = choice(["Determinant", "Multiplication Matrix", "Calculation"])
    if Choice == "Determinant":
        Number = randint(1, 5)
        Matrix = [[choice(Numbers) for j in range(Number)] for i in range(Number)]
        Question = "Solve this: det(" + str(Matrix) + ") = "
        Product = int(input(Question))
        if Product == det(Matrix):
            print("Excellent! 👏")
        return "Starter WON!"
    else:
        print("Sorry kid 👎")
        return "Starter LOST!"

#Calculation of multiplication of two matrices.
elif Choice == "Multiplication Matrix":
    Number_of_the_rows_of_the_first_matrix = randint(1, 5)
    Number_of_the_columns_of_the_first_matrix_and_number_of_rows_of_the_second_matrix = randint(1, 5)
    Number_of_the_columns_of_the_second_matrix = randint(1, 5)
    Matrix1 = [ [ choice(Numbers) for j in range( Number_of_the_columns_of_the_first_matrix_and_number_of_rows_of_the_second_matrix ) ] for i in range( Number_of_the_rows_of_the_first_m
    Matrix2 = [ [ choice(Numbers) for j in range( Number_of_the_columns_of_the_second_matrix ) ] for i in range( Number_of_the_columns_of_the_first_matrix_and_number_of_rows_of_the_sec
    Question = "Solve this: " + str(Matrix1) + " * " + str(Matrix2) + " = "
    Solution = [ [ 0 for j in range( len( Matrix2[0] ) ) ] for i in range( len( Matrix1 ) ) ]
    for i in range( len( Matrix1 ) ):
        for j in range( len( Matrix2[0] ) ):
            for k in range( len( Matrix2 ) ):
                Solution[i][j] += ( Matrix1[i][k] * Matrix2[k][j] )
    Product = eval(input(Question)) #Type(Product) = Matrix(Nested List)
    if Product == Solution:
        print("Excellent! 👏")
        return "Starter WON!"
    else:
        print("Sorry kid 👎")
        return "Starter LOST!"

#Basic Operations
else:
    Operation = choice(["*", "**", "//", "+", "-"])
    if Operation == "*":
        Solution = Number1 * Number2
    elif Operation == "**":
        Solution = Number1 ** Number2
    elif Operation == "//":
        Solution = Number1 // Number2
    elif Operation == "+":
        Solution = Number1 + Number2
    else:
        Solution = Number1 - Number2
    Question = "Solve this: " + str( Number1 ) + " " + Operation + " " + str( Number2 ) + " = "
    Product = int(input(Question))
    print()
    if Product == Solution:
        print("Excellent! 👏")
        return "Starter WON!"
    else:
        print("Sorry kid 👎")
```

return "Starter LOST!"

تابع به صورت تصادفی یکی از موارد زیر را از بازیکن انتخاب کننده می خواهد:

1. دترمینان یک ماتریس مربعی یک در یک، دو در دو... و یا پنج در پنج. (به وسیله زیر تابع ( ) det)
2. ضرب دو ماتریس.
3. حاصل جمع، ضرب، تفریق و تقسیم بین دو عدد.

تابع ( ) det بازگشتی بود و از تابع ( ) cofactor برای محاسبه دترمینان (بنا بر تعریف) بهره می برد.

تابع مختص در دوم

```python
def Door2(Enigmas):
    if Enigmas != {}:
        Question = choice(list(Enigmas.keys()))
        Question_ = Question + "\n"
        Answer = input( Question_ )
        print()
        if Enigmas[Question] == Answer:
            print(choice(["WHAT?! How did you solve that? You must have gotten help from Google especially Youtube..", "It wasn't that hard! Don't put on a supercilious air.", "WHAT? Ok. Se
            return ["Starter WON!", Question]
        else:
            print("WRONG!")
            return ["Starter LOST!", Question]
        #Returning a list exceptionally for to prevent showing an enigma more than one time.

    else:
        #If there's no enigmas to be asked, we'll have to choose other doors.
        newDoor = input("Nothing to be asked. Choose another door. <D1, D3>\n")
        while newDoor not in ["D1", "D3"]:
            newDoor = input("Nothing to be asked. Choose another door. <D1, D3>\n")
        if newDoor == "D1":
            if Door1() == "Starter WON!":
                return "Starter WON!"
            else:
                return "Starter LOST!"
        else:
            if Door3() == "Starter WON!":
                return "Starter WON!"
            else:
                return "Starter LOST!"
```

تابع دیکشنری ای از معماها و جواب آنها را به عنوان ورودی گرفته و جواب بازیکن را با جواب معما مقایسه می کند.

اگر پاسخ صحیح یا غلط بود، به ترتیب مقادیر مربوطه را بر میگرداند.

اگر دیکشنری خالی از سوال جدید برای پرسش باشد، بازیکن موظف است یکی از دو در دیگر را انتخاب کند. در این صورت شرط کاهش دو برابری سلامتی برای بازیکنی که حال مجبور است یکی از دو در را انتخاب کند، در صورت انتخاب درسوم، وجود ندارد.

تابع مختص در سوم

```
def Door3():
    Game=input("Choose a game <Quoridor, Tic-Tac-Toe, Rock Paper Scissors, Flower or nonsense> ")
    while Game not in ["Quoridor", "Tic-Tac-Toe", "Rock Paper Scissors", "Flower or nonsense"]:
        Game=input("Choose a game <Quoridor, Tic-Tac-Toe, Rock Paper Scissors, Flower or nonsense> ")

    if Game == "Quoridor":
        #Quoridor absolutely has a winner!
        Result = Quoridor()
        if Result == "Starter WON!":
            return "Starter WON!"
        else:
            return "Starter LOST!"

    elif Game == "Tic-Tac-Toe":
        Result = TicTacToe()
        #Whenever Draw.
        while Result == "DRAW!":
            Result = TicTacToe()
        if Result == "Starter WON!":
            return "Starter WON!"
        else:
            return "Starter LOST!"

    elif Game == "Rock Paper Scissors":
        Result = RockPaperScissors()
        #Whenever Draw.
        while Result == "DRAW!":
            Result = RockPaperScissors()
        if Result == "Starter WON!":
            return "Starter WON!"
        else:
            return "Starter LOST!"

    else:
        Result = Flower_or_nonsense()
        #Whenever Draw.
        while Result == "DRAW!":
            Result = Flower_or_nonsense()
        if Result == "Starter WON!":
            return "Starter WON!"
        else:
            return "Starter LOST!"
#Each game is defined at next.
```

این تابع حق انتخاب یکی از چهار بازی "کوریدور"، "دوز"، "سنگ کاغذ قیچی" و "گل یا پوچ" را به بازیکن داده و در صورت تساوی در هر یک از این بازی ها(برگردانده شدن مقدار "DRAW!")، بازی تا پیروزی یکی از این دو بازیکن تکرار خواهد شد.

(به جزء بازی کوریدور که هرگز بدون برنده نخواهد بود.)

در ادامه توابعی را به منظور استفاده در فضای بازی، طراحی کرده ایم:

QUORIDOR

```python
#Quoridor
def Quoridor():
    #Making the board of the game without any players.
    Board = ["E.E.E.E.E.E.E.E.E"]
    for Row in range(8):
        Line_of_dots = ""
        Line_of_empty_seats = ""
        for Column in range(9):
            Line_of_dots += ". "
            Line_of_empty_seats += "E."
        Board.append( Line_of_dots[: len( Line_of_dots ) ] )
        Board.append( Line_of_empty_seats[: len( Line_of_empty_seats ) - 1 ] )

    #Function for moving up.
    def Up( Location ):
        #Condition of limitations.
        if Location[0] == 1 or Board[ ( Location[0] - 1 ) * 2 - 1 ][ ( Location[1] - 1 ) * 2 ] == "-":
            print("There's no seat upside! Try the other sides.")
            Side = input("Down, right, or left? < D, R, L > ")
            while Side not in ["D", "R", "L"]:
                Side = input("Down, right, or left? < D, R, L > ")
            while True:
                if Side == "D":
                    Down( Location )
                    break
                elif Side == "R":
                    Right( Location )
                    break
                else:
                    Left( Location )
                    break
        else:
            #If your rival is above you:
            if Board[ ( Location[0] - 2 ) * 2 ][ ( Location[1] - 1 ) * 2 ] != "E":
                if Location[0] == 2 or Board[ ( Location[0] - 2 ) * 2  - 1][ ( Location[1] - 1 ) * 2 ] == "-":
                    Side = input("Right or left? < R, L > ")
                    while Side not in ["R", "L"]:
                        Side = input("Right or left? < R, L > ")
                    if Side == "R":
                        if Location[1] == 9:
                            print("Moving right is not allowed. You'll be brought left automatedly.")
                            Location[0] -= 1
                            Location[1] -= 1
                        else:
                            Location[0] -= 1
                            Location[1] += 1
                else:
                    if Location[1] == 1:
                        print("Moving left is not allowed. You'll be brought right automatedly.")
                        Location[0] -= 1
                        Location[1] += 1
                    else:
                        Location[0] -= 1
                        Location[1] -= 1
            else:
                Location[0] -= 2
        else:
            Location[0] -= 1

#Function for moving down.
def Down( Location ):
    #Condition of limitations.
    if Location[0] == 9 or Board[ ( Location[0] - 1 ) * 2 + 1 ][ ( Location[1] - 1 ) * 2 ] == "-":
        print("There's no seat downside! Try the other sides.")
        Side = input("Up, right, or left? < U, R, L > ")
        while Side not in ["U", "R", "L"]:
            Side = input("Up, right, or left? < U, R, L > ")
        while True:
            if Side == "U":
                Up( Location )
                break
            elif Side == "R":
                Right( Location )
                break
            else:
                Left( Location )
                break
    else:
        #If your rival is under you:
        if Board[ Location[0] * 2 ][ ( Location[1] - 1 ) * 2 ] != "E":
            if Location[0] == 8 or Board[ Location[0] * 2 + 1 ][ ( Location[1] - 1 ) * 2 ] == "-":
                Side = input("Right or left? < R, L > ")
                while Side not in ["R", "L"]:
                    Side = input("Right or left? < R, L > ")
                if Side == "R":
                    if Location[1] == 9:
                        print("Moving right is not allowed. You'll be brought left automatedly.")
                        Location[0] += 1
                        Location[1] -= 1
                    else:
                        Location[0] += 1
                        Location[1] += 1
```

```python
                else:
                    if Location[1] == 1:
                        print("Moving left is not allowed. You'll be brought right automatedly.")
                        Location[0] += 1
                        Location[1] += 1
                    else:
                        Location[0] += 1
                        Location[1] -= 1
            else:
                Location[0] += 2
        else:
            Location[0] += 1

#Function for moving right.
def Right( Location ):
    #Condition of limitations.
    if Location[1] == 9 or Board[ ( Location[0] - 1 ) * 2 ][ ( Location[1] - 1 ) * 2 + 1 ] == "|":
        print("There's no seat rightside! Try the other sides.")
        Side = input("Up, down, or left? < U, D, L > ")
        while Side not in ["U", "D", "L"]:
            Side = input("Up, down, or left? < U, D, L > ")
        while True:
            if Side == "U":
                Up( Location )
                break
            elif Side == "D":
                Down( Location )
                break
            else:
                Left( Location )
                break
    else:
        #If your rival is on your right:
        if Board[ ( Location[0] - 1 ) * 2 ][ ( Location[1] - 1 ) * 2 + 2 ] != "E":
            if Location[1] == 8 or Board[ ( Location[0] - 1 ) * 2 ][ ( Location[1] - 1 )* 2 + 3 ] == "|":
                Side = input("Right or left? < R, L > ")
                while Side not in ["R", "L"]:
                    Side = input("Right or left? < R, L > ")
                if Side == "L":
                    if Location[0] == 1:
                        print("Moving left is not allowed. You'll be brought your rival's right automatedly.")
                        Location[0] += 1
                        Location[1] += 1
                    else:
                        Location[0] -= 1
                        Location[1] += 1
                    else:
                        if Location[0] == 9:
                            print("Moving right is not allowed. You'll be brought your rival's left automatedly.")
                            Location[0] -= 1
                            Location[1] += 1
                        else:
                            Location[0] += 1
                            Location[1] += 1
                else:
                    Location[1] += 2
            else:
                Location[1] += 1

#Function for moving left.
def Left( Location ):
    #Condition of limitations.
    if Location[1] == 1 or Board[ ( Location[0] - 1 ) * 2 ][ ( Location[1] - 1 ) *  2 - 1 ] == "|":
        print("There's no seat leftside! Try the other sides.")
        Side = input("Up, down, or right? < U, D, R > ")
        while Side not in ["U", "D", "R"]:
            Side = input("Up, down, or right? < U, D, R > ")
        while True:
            if Side == "U":
                Up( Location )
                break
            elif Side == "D":
                Down( Location )
                break
            else:
                Right( Location )
                break
    else:
        #If your rival is on your left:
        if Board[ ( Location[0] - 1 ) * 2 ][ ( Location[1] - 1 ) * 2 - 2 ] != "E":
            if Location[1] == 2 or Board[ ( Location[0] - 1 ) * 2 ][ ( Location[1] - 1 ) * 2 - 3 ] == "|":
                Side = input("Right or left? < R, L > ")
                while Side not in ["R", "L"]:
                    Side = input("Right or left? < R, L > ")
                if Side == "L":
                    if Location[0] == 1:
                        print("Moving left is not allowed. You'll be brought your rival's right automatedly.")
                        Location[0] += 1
                        Location[1] -= 1
                    else:
                        Location[0] -= 1
                        Location[1] -= 1
                else:
                    if Location[0] == 8:
                        print("Moving right is not allowed. You'll be brought your rival's left automatedly.")
                        Location[0] -= 1
                        Location[1] -= 1
                    else:
                        Location[0] += 1
                        Location[1] -= 1
            else:
                Location[1] -= 2
        else:
            Location[1] -= 1

#The function helps the player to move.
def Move( Location ):
    #Selecting the side for moving at least for the first time.
    Side = input("Up, down, right, or left? < U, D, R, L > " )
    while Side not in ["U", "D", "R", "L"]:
        Side = input("Up, down, right, or left? < U, D, R, L > " )
    while True:
        if Side == "U":
            Up( Location )
            break

        elif Side == "D":
            Down( Location )
            break

        elif Side == "R":
            Right( Location )
            break
        else:
            Left( Location )
            break
```

توابعی را به منظور تعریف جهت برای بازی کوریدور تعریف کردیم که با نام مربوطه تشکیل شده اند.

چنانچه قادر به حرکت در جهتی نباشیم، تابع آن جهت بازیکن را قادر می سازد تا جهات دیگری را انتخاب نماید.

تابع Move ( ) به بازیکن اجازه میدهد تا اولین حرکت خود را آغاز کند.

```
#The function helps the player to place the walls.
def WallPicker():
    #Selecting the direction.
    Direction = input("Vertical of horizontal? < V or H > ")
    while Direction not in ["V", "H"]:
        Direction = input("I said: 'Vertical of horizontal?' < V or H > ")

    if Direction == "V":
        #Selected row and next to it on its right will be occupied.
        Row = int(input("Which row? < 1 - 8 > "))
        while Row > 8 or Row < 1:
            Row = int(input("Out of range! Which row? < 1 - 8 > "))
        Column = int(input("Which column? < 1 - 8 > "))
        while Column > 8 or Column < 1:
            Column = int(input("Out of range! Which column? < 1 - 8 > "))
        while Board[ ( Row - 1 ) * 2 ][ Column * 2 - 1 ] != ".":
            print("You can't place a wall on a wall! Try again.")
            Row = int(input("Which row? < 1 - 8 > "))
            while Row > 8 or Row < 1:
                Row = int(input("Out of range! Which row? < 1 - 8 > "))
            Column = int(input("Which column? < 1 - 8 > "))
            while Column > 8 or Column < 1:
                Column = int(input("Out of range! Which column? < 1 - 8 > "))
        else:
            Board[ ( Row - 1 ) * 2 ] = Board[ ( Row - 1 ) * 2 ][: Column * 2 - 1 ] + "|" + Board[ ( Row - 1 ) * 2 ][ Column * 2 :]
            Board[ ( Row - 1 ) * 2 + 2 ] = Board[ ( Row - 1 ) * 2 + 2 ][: Column * 2 - 1 ] + "|" + Board[ ( Row - 1 ) * 2 + 2 ][ Column * 2 :]

    else:
        Row = int(input("Which row? < 1 - 8 > "))
        while Row not in range(1, 9):
            Row = int(input("Out of range! Which row? < 1 - 8 > "))
        #Selected column and next to it on its right will be occupied.
        Column = int(input("Which column? < 1 - 8 > "))
        while Column not in range(1, 9):
            Column = int(input("Out of range! Which column? < 1 - 8 > "))
        while Board[ Row * 2 - 1 ][( Column - 1 ) * 2 ] == "-" or Board[ Row * 2 - 1 ][ Column * 2 ] == "-":
            print("You can't place a wall on a wall! Try again.")
        Row = int(input("Which row? < 1 - 8 > "))
        while Row not in range(1, 9):
            Row = int(input("Out of range! Which row? < 1 - 8 > "))
        Column = int(input("Which column? < 1 - 8 > "))
        while Column not in range(1, 9):
            Column = int(input("Out of range! Which column? < 1 - 8 > "))
        Board[ Row * 2 - 1 ] = Board[ Row * 2 - 1 ][: ( Column - 1 ) * 2 ] + "- -" + Board[ Row * 2 - 1 ][ Column * 2 + 1 :]
```

تابع ( ) WallPicker بازیکن را قادر می سازد تا در مختصاتی مشخص، دیوار قرار دهد.

دیوارها را می توان به صورت افقی (Horizontal)

یا عمودی (Vertical) قرار داد.

در صورتی که در مختصات منتخب از قبل دیواری قرار گرفته باشد، تابع به بازیکن اجازه می دهد تا مختصاتی دیگر اختیار کند.

حال تابع شروع بازی کوریدور را به صورت زیر می سازیم:

```python
def Start():
    #Preparing starting atmosphere.
    Location1 = [1, 5]
    Location2 = [9, 5]

    #              Col
    #               |
    #               v
    #Location = [i, j]
    #            ^
    #            |
    #           Row

    #Each player have ten walls at first.
    Number_of_walls1 = 10
    Number_of_walls2 = 10

    Line_of_walls1 = "| | | | | | | | | |"
    Line_of_walls2 = "| | | | | | | | | |"

    Board[ ( Location1[0] - 1 ) * 2 ] = Board[ ( Location1[0] - 1 ) * 2 ][: ( Location1[1] - 1 ) * 2 ] + "1" + Board[ ( Location1[0] - 1 ) * 2 ][ ( Location1[1] - 1 ) * 2 + 1 :]
    Board[ ( Location2[0] - 1 ) * 2 ] = Board[ ( Location2[0] - 1 ) * 2 ][: ( Location2[1] - 1 ) * 2 ] + "2" + Board[ ( Location2[0] - 1 ) * 2 ][ ( Location2[1] - 1 ) * 2 + 1 :]

    print()
    print("Player1's walls.")
    print(Line_of_walls1)
    print()
    for Element in Board:
        print(Element)
    print()
    print("Player2's walls.")
    print(Line_of_walls2)
```

بازی را با قرار دادن مهره های دو بازیکن در وسط ردیف های اول و آخر و لحاظ کردن 10 دیوار برای هر یک از آنها، آغاز می کنیم.

```
while True:
    #Player one's turn.
    print()
    Choice = input("Player1. Move or Wall? < M or W > ")
    while Choice not in ["M", "W"]:
        Choice = input("I said: 'Move or Wall?' < M or W > ")
    if Choice == "M" or Number_of_walls1 == 0:
        if Number_of_walls1 == 0:
            print("Player 1. You have no wall to place. Just move!")

        #Cleaning player one's footprint.
        Board[ ( Location1[0] - 1 ) * 2 ] = Board[ ( Location1[0] - 1 ) * 2 ][: ( Location1[1] - 1 ) * 2 ] + "E" + Board[ ( Location1[0] - 1 ) * 2 ][ ( Location1[1] - 1 ) * 2 + 1 :]

        #It's time to move.
        Move( Location1 )

        #Locating player1.
        Board[ ( Location1[0] - 1 ) * 2 ] = Board[ ( Location1[0] - 1 ) * 2 ][: ( Location1[1] - 1 ) * 2 ] + "1" + Board[ ( Location1[0] - 1 ) * 2 ][ ( Location1[1] - 1 ) * 2 + 1 :]

    else:
        #While the player have some walls, function 'Wall_picker()' will be called.
        if Number_of_walls1 == 1:
            print("You have only a wall.")
        else:
            print("You have", Number_of_walls1, "walls.")
        WallPicker()
        Number_of_walls1 -= 1
        for Element in Line_of_walls1:
            if Element == "|":
                Line_of_walls1 = Line_of_walls1[: Line_of_walls1.find( Element ) ] + "X" + Line_of_walls1[ Line_of_walls1.find( Element ) + 1 :]
                break
    print()
    print("Player1's walls.")
    print(Line_of_walls1)
    print()
    for Element in Board:
        print(Element)
    print()
    print("Player2's walls.")
    print(Line_of_walls2)

    #Winning condition for Player1.
    if Location1[0] == 9:
        print()
        print("Player1 wins!")
        return "Starter WON!"
        break
```

```
#Player two's turn.
print()
Choice = input("Player2. Move or Wall? < M or W > ")
while Choice not in ["M", "W"]:
    Choice = input("I said: 'Move or Wall?' < M or W > ")
if Choice == "M" or Number_of_walls2 == 0:
    if Number_of_walls2 == 0:
        print("Player2. You have no wall to place. Just move!")

    #Cleaning player two's footprint.
    Board[ ( Location2[0] - 1 ) * 2 ] = Board[ ( Location2[0] - 1 ) * 2 ][: ( Location2[1] - 1 ) * 2 ] + "E" + Board[ ( Location2[0] - 1 ) * 2 ][ ( Location2[1] - 1 ) * 2 + 1 :]

    #It's time to move.
    Move( Location2 )

    #Locating player2.
    Board[ ( Location2[0] - 1 ) * 2 ] = Board[ ( Location2[0] - 1 ) * 2 ][: ( Location2[1] - 1 ) * 2 ] + "2" + Board[ ( Location2[0] - 1 ) * 2 ][ ( Location2[1] - 1 ) * 2 + 1 :]

else:
    #While the player have some walls, function 'Wall_picker()' will be called.
    if Number_of_walls2 == 1:
        print("You have only a wall.")
    else:
        print("You have", Number_of_walls2, "walls.")
    WallPicker()
    Number_of_walls2 -= 1
    for Element in Line_of_walls2:
        if Element == "|":
            Line_of_walls2 = Line_of_walls2[: Line_of_walls2.find( Element ) ] + "X" + Line_of_walls2[ Line_of_walls2.find( Element ) + 1 :]
            break
print()
print("Player1's walls.")
print(Line_of_walls1)
print()
for Element in Board:
    print(Element)
print()
print("Player2's walls.")
print(Line_of_walls2)

#Winning condition for Player2.
if Location2[0] == 1:
    print()
    print("Player2 wins!")
    return "Starter LOST!"
    break
```

Start()

حلقه بی نهایتی را تشکیل می دهیم تا نوبت بازیکن ها تا شرط پیروزی هر یک از آنها، به صورت پویا تکرار شود.

اگرهر یک از بازیکن ها زودتر به آخرین ردیف مقابل نسبت به موقعیت مکانی شروع خود رسد، بازی به نفع او به پایان می رسد.

Tic-Tac-Toe

```
#Tic-Tac-Toe
#Starter & its opponent's symbols are 'X' & 'O' respectively.
def TicTacToe():
    #The function helps us do our job.
    def Placing_the_shape( Board, Symbol ):
        Row = int(input("Which row do you want to place your symbol? < 1 - 3 > "))
        while Row not in range(1, 4):
            Row = int(input("Which row do you want to place your symbol? < 1 - 3 > "))
        Column = int(input("Which column do you want to place your symbol? < 1 - 3 > "))
        while Column not in range(1, 4):
            Column = int(input("Which column do you want to place your symbol? < 1 - 3 > "))
        while " " not in Board[ ( Row - 1 ) * 3 + 1 ][ ( Column - 1 ) * 6 + 2 ]:
            Row = int(input("Which row do you want to place your symbol? < 1 - 3 > "))
            while Row not in range(1, 4):
                Row = int(input("Which row do you want to place your symbol? < 1 - 3 > "))
            Column = int(input("Which column do you want to place your symbol? < 1 - 3 > "))
            while Column not in range(1, 4):
                Column = int(input("Which column do you want to place your symbol? < 1 - 3 > "))
        Board[ ( Row - 1 ) * 3 + 1 ] = Board[ ( Row - 1 ) * 3 + 1 ][: ( Column - 1 ) * 6 + 2 ] + Symbol + Board[ ( Row - 1 ) * 3 + 1 ][ ( Column - 1 ) * 6 + 3 :]


    #The function which starts the game.
    def Start():
        print()
        for Element in Board:
            print(Element)
        Equality = 0
        Breaker = False
        while True:
            #Player one's turn.
            print()
            print("PLAYER-1, your turn.")
            Placing_the_shape( Board,"X" )
            for Element in Board:
                print(Element)

            #Winning conditions for PLAYER1.
            for Row in range(1, 4):
                if all( Board[ ( Row - 1 ) * 3 + 1 ][ ( Column - 1 ) * 6 + 2 ] == "X" for Column in range(1, 4) ):
                    Breaker = True
                    break
            for Column in range(1, 4):
                if all( Board[ ( Row - 1 ) * 3 + 1 ][ ( Column - 1 ) * 6 + 2 ] == "X" for Row in range(1, 4) ):
                    Breaker = True
                    break
        #Checking diagonal sides.
        if all( Board[ ( Place - 1 ) * 3 + 1 ][ ( Place - 1 ) * 6 + 2 ] == "X" for Place in range(1, 4) ) or all( Board[ ( Place - 1 ) * 3 + 1 ][ ( 1 - Place ) * 6 + 14 ] == "X" for Pla
            Breaker = True

        if Breaker:
            print("PLAYER-1 wins!")
            return "Starter WON!"
            break

        #If none of the conditions is satisfied, the game will be continued...
        Equality += 1

        #Equality condition.
        if Equality == 9:
            print("Perfectly Balanced!")
            break

        #Player two's turn.
        print()
        print("PLAYER-2, your turn.")
        Placing_the_shape( Board,"O" )
        for Element in Board:
            print(Element)

        #Winning conditions for PLAYER2.
        for Row in range(1, 4):
            if all( Board[ ( Row - 1 ) * 3 + 1 ][ ( Column - 1 ) * 6 + 2 ] == "O" for Column in range(1, 4) ):
                Breaker = True
                break
        for Column in range(1, 4):
            if all( Board[ ( Row - 1 ) * 3 + 1 ][ ( Column - 1 ) * 6 + 2 ] == "O" for Row in range(1, 4) ):
                Breaker = True
                break
        #Checking diagonal sides.
        if all( Board[ ( Place - 1 ) * 3 + 1 ][ ( Place - 1 ) * 6 + 2 ] == "O" for Place in range(1, 4) ) or all( Board[ ( Place - 1 ) * 3 + 1 ][ ( 1 - Place ) * 6 + 14 ] == "O" for Pla
            Breaker = True

        if Breaker:
            print("PLAYER-2 wins!")
            return "Starter LOST!"
            break

        #If none of the conditions is satisfied, the game will be continued...
        Equality += 1

Space_for_symbols = " "
Underline_for_symbols = "_"
```

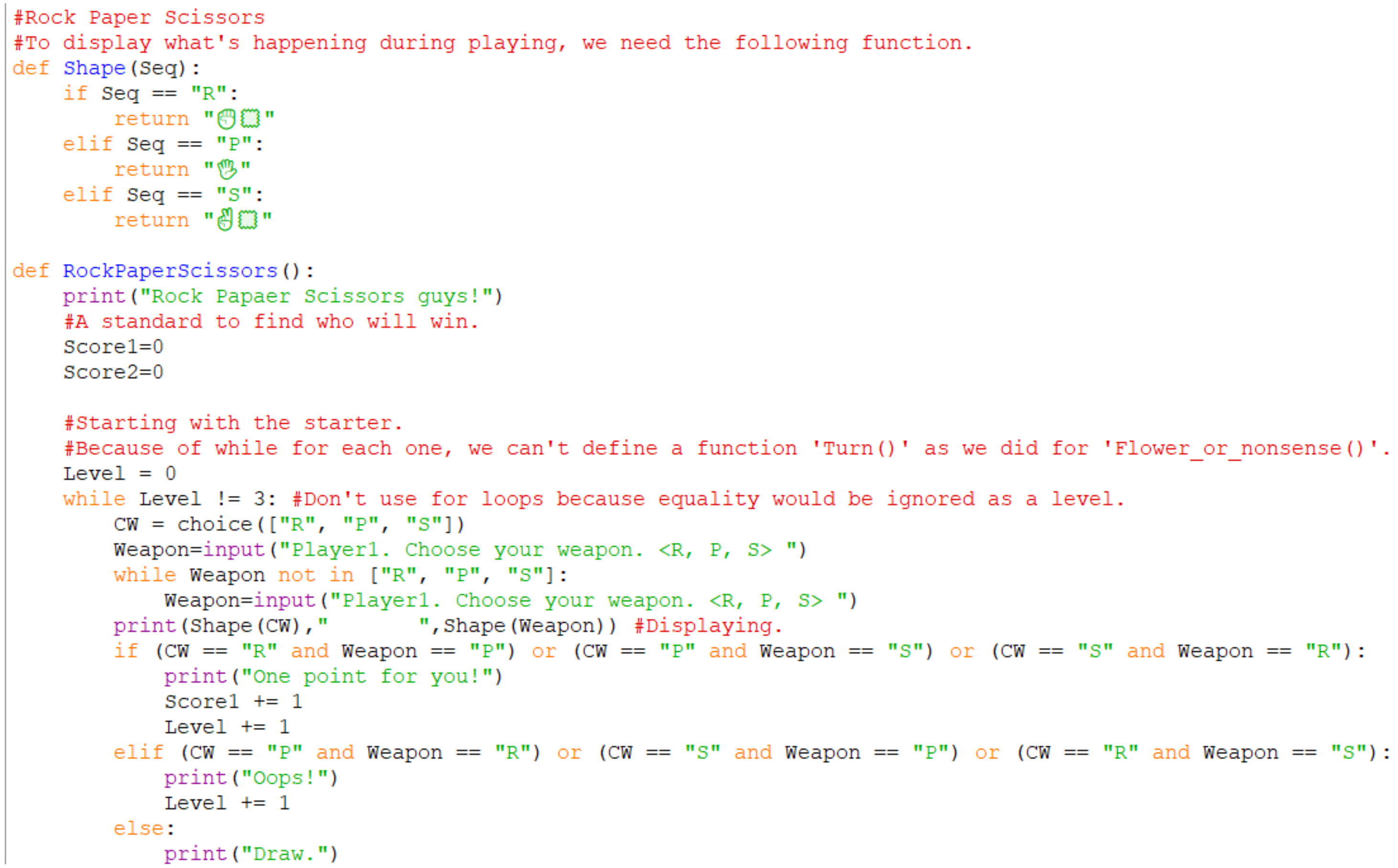
```
Board = ["  " + Space_for_symbols + "  |  " + Space_for_symbols + "  |  " + Space_for_symbols + "  ",
        "  " + Space_for_symbols + "  |  " + Space_for_symbols + "  |  " + Space_for_symbols + "  ",
        "__" + Underline_for_symbols + "__|__" + Underline_for_symbols + "__|__" + Underline_for_symbols + "__",
        "  " + Space_for_symbols + "  |  " + Space_for_symbols + "  |  " + Space_for_symbols + "  ",
        "  " + Space_for_symbols + "  |  " + Space_for_symbols + "  |  " + Space_for_symbols + "  ",
        "__" + Underline_for_symbols + "__|__" + Underline_for_symbols + "__|__" + Underline_for_symbols + "__",
        "  " + Space_for_symbols + "  |  " + Space_for_symbols + "  |  " + Space_for_symbols + "  ",
        "  " + Space_for_symbols + "  |  " + Space_for_symbols + "  |  " + Space_for_symbols + "  ",
        "  " + Space_for_symbols + "  |  " + Space_for_symbols + "  |  " + Space_for_symbols + "  "]
Score1 = 0
Score2 = 0

#Starting for one time; so we can use 'elif'!
Result = Start()
print()
if Result == "Starter WON!":
    Score1 += 1
elif Result == "Starter LOST!":
    Score2 += 1
if Score1 > Score2:
    return "Starter WON!"
elif Score1 < Score2:
    return "Starter LOST!"
else:
    #Whenever equality, match again!
    print("Fight again!")
    print("------------")
    return "DRAW!"
```

تابع ( )Placing_the_shape با گرفتن شکل و صفحه بازی از ورودی، شکل بازیکن را در مختصاتی مشخص از صفحه قرار می دهد.

تابع بازی ()Start حاوی یک حلقه بی نهایت است و این حلقه تا زمانی اجرا می شود که یکی از اشکال یک ردیف، یک ستون و یا یک قطر را در بر گیرد. در غیر اینصورت و با تکمیل شدن ظرفیت جدول، بازی با نتیجه مساوی به پایان خواهد رسید.

Rock Paper Scissors



```
#Rock Paper Scissors
#To display what's happening during playing, we need the following function.
def Shape(Seq):
    if Seq == "R":
        return "✊⬚"
    elif Seq == "P":
        return "✋"
    elif Seq == "S":
        return "✌⬚"

def RockPaperScissors():
    print("Rock Papaer Scissors guys!")
    #A standard to find who will win.
    Score1=0
    Score2=0

    #Starting with the starter.
    #Because of while for each one, we can't define a function 'Turn()' as we did for 'Flower_or_nonsense()'.
    Level = 0
    while Level != 3: #Don't use for loops because equality would be ignored as a level.
        CW = choice(["R", "P", "S"])
        Weapon=input("Player1. Choose your weapon. <R, P, S> ")
        while Weapon not in ["R", "P", "S"]:
            Weapon=input("Player1. Choose your weapon. <R, P, S> ")
        print(Shape(CW),"       ",Shape(Weapon)) #Displaying.
        if (CW == "R" and Weapon == "P") or (CW == "P" and Weapon == "S") or (CW == "S" and Weapon == "R"):
            print("One point for you!")
            Score1 += 1
            Level += 1
        elif (CW == "P" and Weapon == "R") or (CW == "S" and Weapon == "P") or (CW == "R" and Weapon == "S"):
            print("Oops!")
            Level += 1
        else:
            print("Draw.")
```

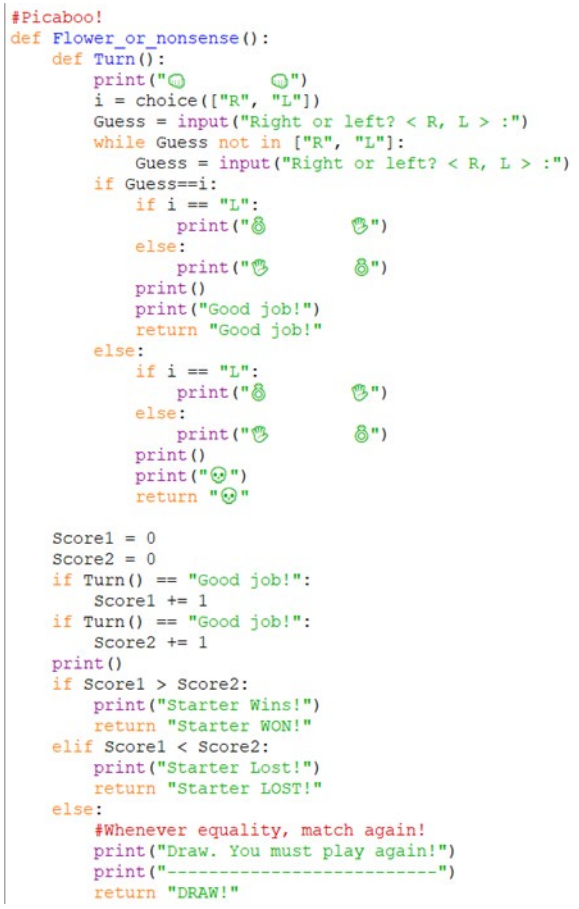
```python
Level = 0
while Level != 3:
    CW = choice(["R", "P", "S"])
    Weapon=input("Player2. Choose your weapon. <R, P, S> ")
    while Weapon not in ["R", "P", "S"]:
        Weapon=input("Player2. Choose your weapon. <R, P, S> ")
    print(Shape(CW),"       ",Shape(Weapon)) #Displaying.
    if (CW == "R" and Weapon == "P") or (CW == "P" and Weapon == "S") or (CW == "S" and Weapon == "R"):
        print("One point for you!")
        Score2 += 1
        Level += 1
    elif (CW == "P" and Weapon == "R") or (CW == "S" and Weapon == "P") or (CW == "R" and Weapon == "S"):
        print("Oops!")
        Level += 1
    else:
        print("Draw.")
print()
if Score1 > Score2:
    print("Starter wins!")
    return "Starter WON!"
elif Score1 < Score2:
    print("Starter lost!")
    return "Starter LOST!"
else:
    #Whenever equality, match again!
    print("Challenge together again!")
    print("-------------------------")
    return "DRAW!"
```

تابع ()Shape برای مزین کرده صفحه به روند بازی به صورت بصری تشکیل شده است.

هر بازیکن در جدال با کامپیوتر، تا سه مرحله سنگ کاغذ قیچی بازی کرده و هر کدام که امتیاز بیشتری کسب نماید، پیروز خواهد بود و در غیر آن صورت، بازی بی نتیجه بوده و از دوباره شروع خواهد شد.

(طبق تابع ()Door3)

Flower or nonsense



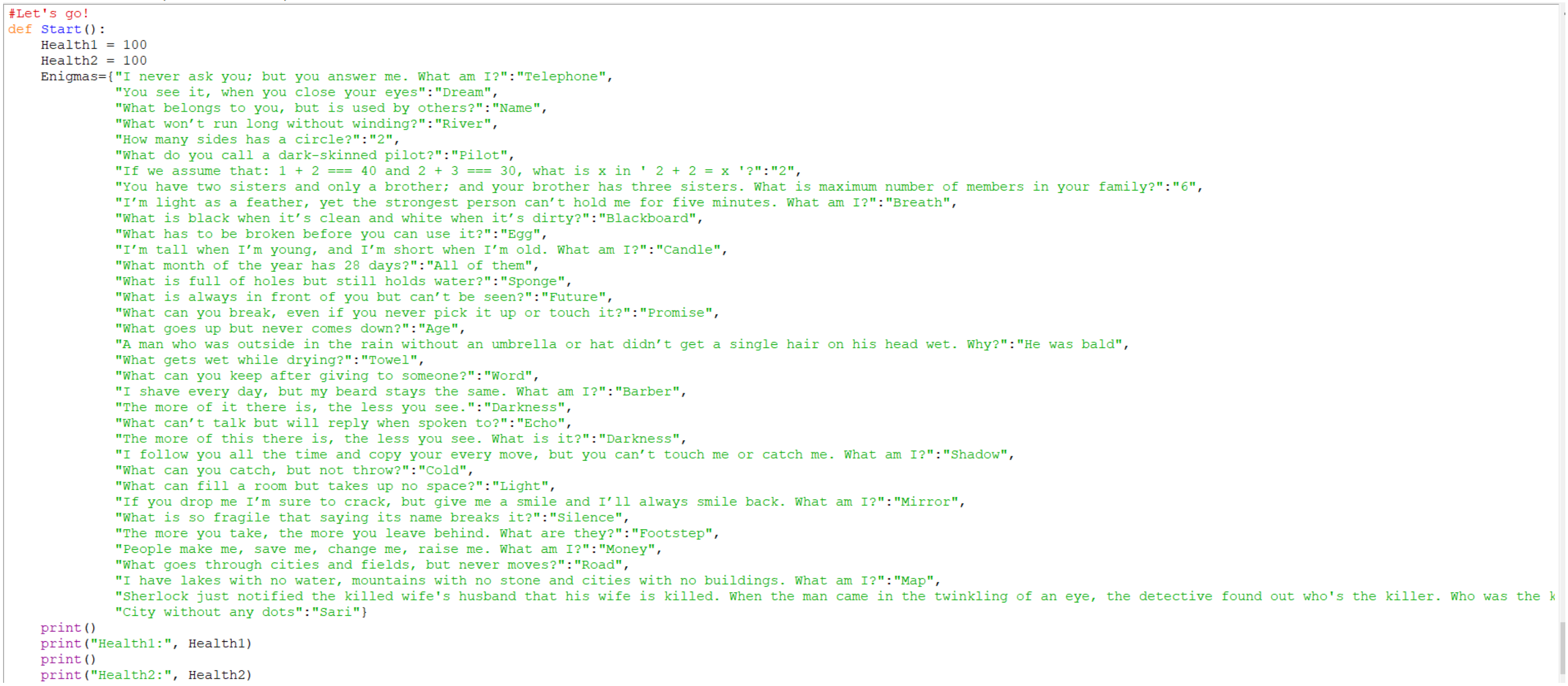
```
#Picaboo!
def Flower_or_nonsense():
    def Turn():
        print("👊         👊")
        i = choice(["R", "L"])
        Guess = input("Right or left? < R, L > :")
        while Guess not in ["R", "L"]:
            Guess = input("Right or left? < R, L > :")
        if Guess==i:
            if i == "L":
                print("💍         ✋")
            else:
                print("✋         💍")
            print()
            print("Good job!")
            return "Good job!"
        else:
            if i == "L":
                print("💍         ✋")
            else:
                print("✋         💍")
            print()
            print("💀")
            return "💀"

    Score1 = 0
    Score2 = 0
    if Turn() == "Good job!":
        Score1 += 1
    if Turn() == "Good job!":
        Score2 += 1
    print()
    if Score1 > Score2:
        print("Starter Wins!")
        return "Starter WON!"
    elif Score1 < Score2:
        print("Starter Lost!")
        return "Starter LOST!"
    else:
        #Whenever equality, match again!
        print("Draw. You must play again!")
        print("--------------------------")
        return "DRAW!"
```

تابع ()Turn نوبت را برای هر یک از بازیکن ها تعبیه می کند.

هر بازیکن با کامپیوتر به چالش کشیده شده و نتیجه مانند بازی سنگ کاغذ قیچی، بررسی می شود.

حال بازی اصلی را به صورت تابعی شامل تمامی توابع مذکور، می سازیم.



```
#Let's go!
def Start():
    Health1 = 100
    Health2 = 100
    Enigmas={"I never ask you; but you answer me. What am I?":"Telephone",
             "You see it, when you close your eyes":"Dream",
             "What belongs to you, but is used by others?":"Name",
             "What won’t run long without winding?":"River",
             "How many sides has a circle?":"2",
             "What do you call a dark-skinned pilot?":"Pilot",
             "If we assume that: 1 + 2 === 40 and 2 + 3 === 30, what is x in ' 2 + 2 = x '?":"2",
             "You have two sisters and only a brother; and your brother has three sisters. What is maximum number of members in your family?":"6",
             "I’m light as a feather, yet the strongest person can’t hold me for five minutes. What am I?":"Breath",
             "What is black when it’s clean and white when it’s dirty?":"Blackboard",
             "What has to be broken before you can use it?":"Egg",
             "I’m tall when I’m young, and I’m short when I’m old. What am I?":"Candle",
             "What month of the year has 28 days?":"All of them",
             "What is full of holes but still holds water?":"Sponge",
             "What is always in front of you but can’t be seen?":"Future",
             "What can you break, even if you never pick it up or touch it?":"Promise",
             "What goes up but never comes down?":"Age",
             "A man who was outside in the rain without an umbrella or hat didn’t get a single hair on his head wet. Why?":"He was bald",
             "What gets wet while drying?":"Towel",
             "What can you keep after giving to someone?":"Word",
             "I shave every day, but my beard stays the same. What am I?":"Barber",
             "The more of it there is, the less you see.":"Darkness",
             "What can’t talk but will reply when spoken to?":"Echo",
             "The more of this there is, the less you see. What is it?":"Darkness",
             "I follow you all the time and copy your every move, but you can’t touch me or catch me. What am I?":"Shadow",
             "What can you catch, but not throw?":"Cold",
             "What can fill a room but takes up no space?":"Light",
             "If you drop me I’m sure to crack, but give me a smile and I’ll always smile back. What am I?":"Mirror",
             "What is so fragile that saying its name breaks it?":"Silence",
             "The more you take, the more you leave behind. What are they?":"Footstep",
             "People make me, save me, change me, raise me. What am I?":"Money",
             "What goes through cities and fields, but never moves?":"Road",
             "I have lakes with no water, mountains with no stone and cities with no buildings. What am I?":"Map",
             "Sherlock just notified the killed wife's husband that his wife is killed. When the man came in the twinkling of an eye, the detective found out who's the killer. Who was the k
             "City without any dots":"Sari"}
    print()
    print("Health1:", Health1)
    print()
    print("Health2:", Health2)
```

بازی را با مقدار دهی سلامتی بازیکنان با مقدار 100 واحد و تشکیل دیکشنری ای از کلیدهای معما و ارزش های جواب آنها، آغاز می کنیم.

```python
while True:
    #Player1's turn.
    print()
    Door = input("PLAYER1. OPEN ONE OF THE DOORS! <D1, D2, D3>\n")
    while Door not in ["D1", "D2", "D3"]:
        Door = input("I SAID:'O-PEN ONE OF THE DOORS!' <D1, D2, D3>\n")
    if Door == "D1":
        if Door1() == "Starter WON!":
            Health2 -= 10
        else:
            Health1 -= 10
    elif Door == "D2":
        Question = Door2(Enigmas)
        if Question[0] == "Starter WON!":
            Health2 -= 10
            Enigmas.pop(Question[1])
        else:
            Health1 -= 10
    else:
        if Door3() == "Starter WON!":
            Health2 -= 10
        else:
            Health1 -= 20
    print()
    print("Health1:", Health1)
    print()
    print("Health2:", Health2)

    #Winning conditions.
    if Health2 == 0:
        print()
        print("PLAYER1 WINS!")
        break

    elif Health1 == 0:
        print()
        print("PLAYER2 WINS!")
        break
        #Player2's turn.
        print()
        Door = input("PLAYER2. OPEN ONE OF THE DOORS! <D1, D2, D3>\n")
        while Door not in ["D1", "D2", "D3"]:
            Door = input("I SAID:'O-PEN ONE OF THE DOORS!' <D1, D2, D3>\n")
        if Door == "D1":
            if Door1() == "Starter WON!":
                Health1 -= 10
            else:
                Health2 -= 10
        elif Door == "D2":
            Question = Door2(Enigmas)
            if Question[0] == "Starter WON!":
                Health2 -= 10
                Enigmas.pop(Question[1])
            else:
                Health1 -= 10
        else:
            if Door3() == "Starter WON!":
                Health1 -= 10
            else:
                Health2 -= 20
        print()
        print("Health1:", Health1)
        print()
        print("Health2:", Health2)

        #Winning conditions.
        if Health2 <= 0:
            print()
            print("PLAYER1 WINS!")
            break

        elif Health1 <= 0:
            print()
            print("PLAYER2 WINS!")
            break
#Playing for the first time.
Start()
```

حلقه بی نهایتی را به منظور انجام بازی در هر نوبت با بررسی شروع برد تشکیل می دهیم.

اگر بازیکن در اول را انتخاب کند و طبق مقدار خروجی تابع در اول، پیروز شود، از سلامتی حریف خود 10 واحد سلامتی کم خواهد شد.

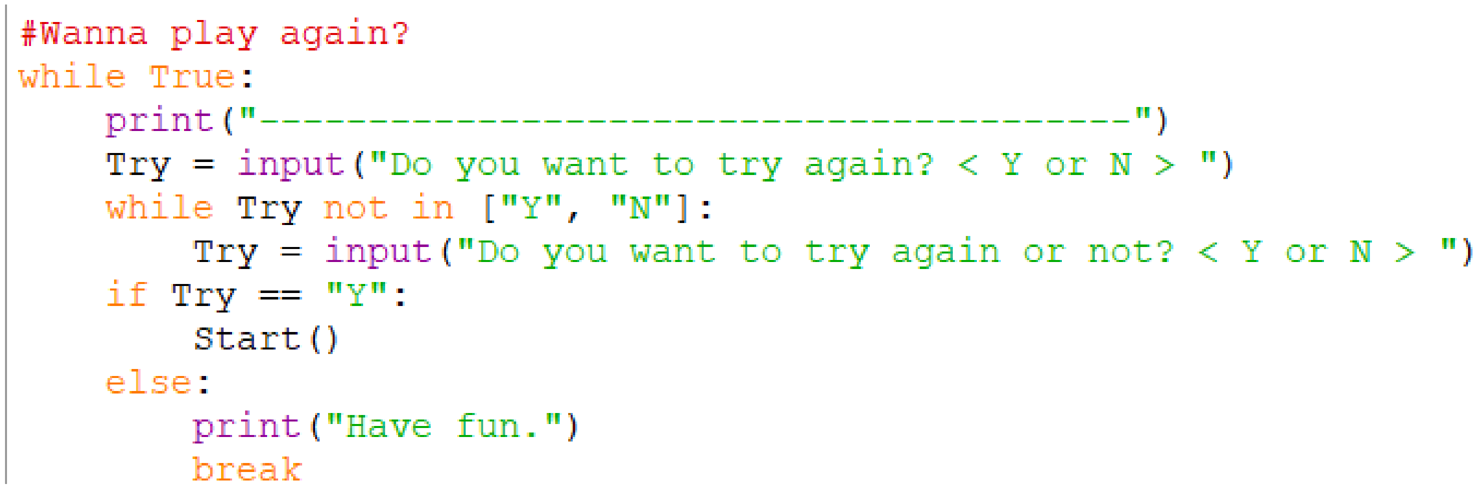
اگر بازیکن در دوم را انتخاب نماید و در حل معما موفق شود، علاوه بر کاهش 10 واحدی سلامتی حریف، برای جلوگیری از تکرار مشاهده معمای حل شده،

معما از دیکشنری اصلی (Enigmas) حذف خواهد شد.

اگر بازیکن در سوم را انتخاب کرده و در بازی ای ببرد یا ببازد،

به ترتیب 10 و 20 واحد از سلامتی حریفش و خود کاسته خواهد شد.

بازیکنی که سلامتی اش زودتر به صفر رسد، بازنده بوده و رقیب او، برنده خواهد شد.



```python
#Wanna play again?
while True:
    print("----------------------------------------")
    Try = input("Do you want to try again? < Y or N > ")
    while Try not in ["Y", "N"]:
        Try = input("Do you want to try again or not? < Y or N > ")
    if Try == "Y":
        Start()
    else:
        print("Have fun.")
        break
```

در آخر نیز برای بازیکنان این قابلیت قرار داده شده است تا مجددا بازی اصلی را انجام دهند.

در این دنیای وسیع برنامه نویسی، هر حرکت، آموزنده است.

از استفاده از دستور ()print گرفته تا استفاده از کلاس های متعدد و ...

برنامه نویسی، تلفیقی از علم نظری و علم تجربی است؛ بطوریکه با منطق های ریاضی، تجربه می آموزید.

همانگونه که من با ساخت یک بازی با قوانین ریاضیات، بسیار آموختم، می آموزم، و خواهم آموخت.

پایان