Cairo University

Faculty of Computers and Artificial Intelligence **CS361**

Artificial Intelligence

2nd Semester 2020 Project

Topic 2 – Gamification

PID: PID23867711

ID	Name	Email
20170341	يوسف اسامه سيد فهمى شحاته القاضى	yousefosama1998@gmail.com
20170074	بانسيه يوسف ابو العباس يوسف	pansyyusuf@gmail.com
20170109	سارة محمود عبد الستار مصطفى	sarashaarawy6@gmail.com

Contents

A. Introduction

Gamification in general refers to the application of game design rules and principles to real/imaginary problems (non-game elements). The concept came from the need of engaging people in different types of activity and dealing with an everlasting problem which lies in boredom and the inability to focus on a problem or a task for large chunks of time. So gamification kicks in with its rules to make any work more fun and engaging. Gamification consists of many elements some of them are:

Avatars: giving the user a certain game like image or icon to represent him in the game.

Score: giving the user a numerical calculation of his performance based on certain criteria (game-like).

Leader boards: giving the user an indication about his performance against other colleagues (enemies).

Story: giving a user an engaging story to explore like any game.

Teammates: giving the user the ability to have allies to help him and achieve their goal together.

These elements are applicable to real life teams and their daily work problems as they have more coordination (teammate element) and have a problem to tackle based on criteria (score, story and leader board elements). So in general, gamifying a workplace or a team flow gives its individuals to express themselves in a way that resembles the fun found in playing video games in addition to finishing their tasks and work as efficient as any other way … maybe more efficient actually!

(References are in the Appendix section of this report).

B. Background

In general, these game elements have a huge resemblance to real life applications. In a game, the player has Health; certain powers and skill; allies to help; enemies to face. In addition, the player has a story to play through and certain missions and tasks to do to help him advance and buy better equipment.

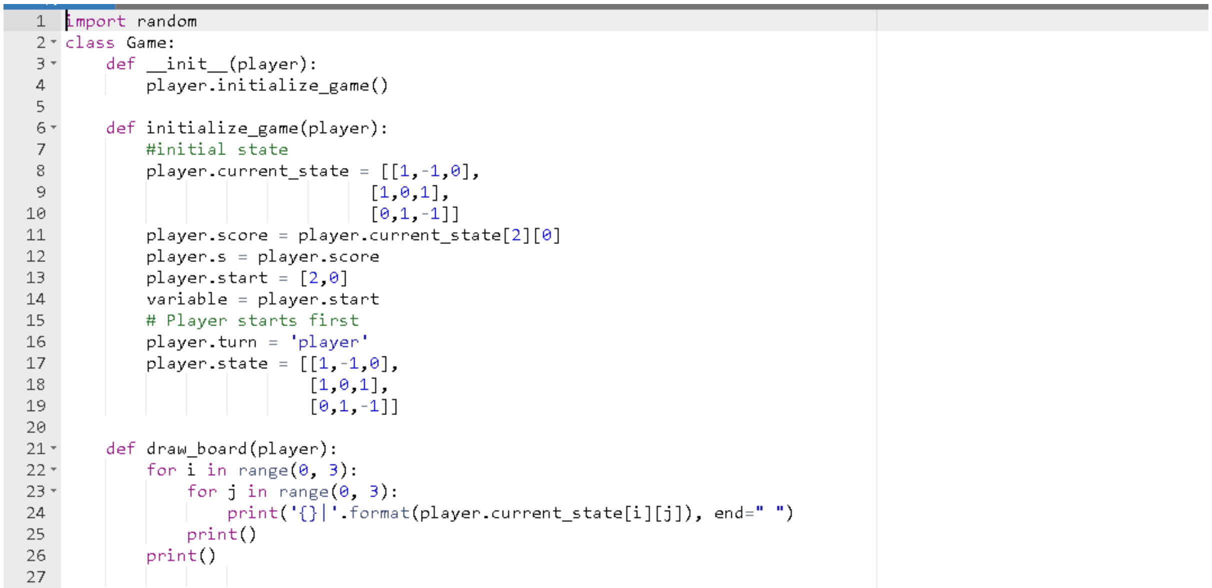
Applying the above principles to real life ... A person has health to care of; physical strength and skills that he can do better than others (cooking, smarts, etc..); friends and family that help him and others who defy him and try to stop him from progressing in life. Also he needs to learn and start a career doing certain tasks and facing problems to help him have a stable income and be able to buy his life needs like food and clothes.

So in short, mapping the principles to each other we get similarity that we talked about above.

Calculating ones' tasks and performance based on numbers helps the person identify his influence and his ability to compete with his peers and fellow workers.

C. Development

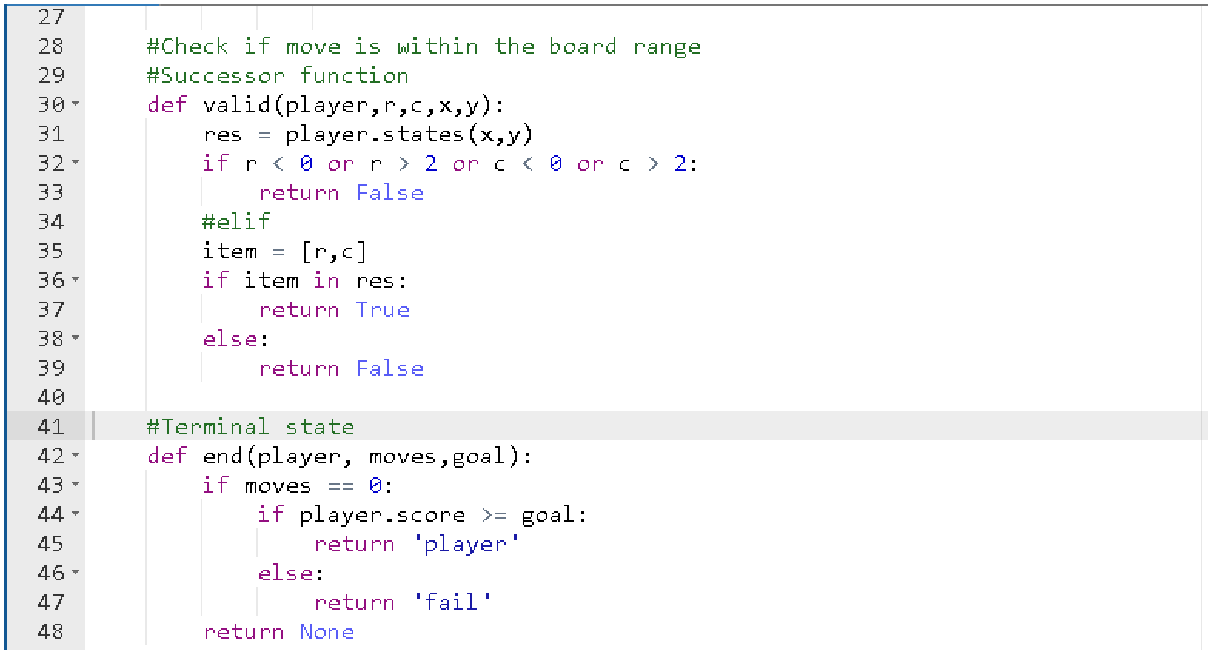
I. System components:



```
import random
class Game:
    def __init__(player):
        player.initialize_game()

    def initialize_game(player):
        #initial state
        player.current_state = [[1,-1,0],
                              [1,0,1],
                              [0,1,-1]]
        player.score = player.current_state[2][0]
        player.s = player.score
        player.start = [2,0]
        variable = player.start
        # Player starts first
        player.turn = 'player'
        player.state = [[1,-1,0],
                        [1,0,1],
                        [0,1,-1]]

    def draw_board(player):
        for i in range(0, 3):
            for j in range(0, 3):
                print('{}|'.format(player.current_state[i][j]), end=" ")
            print()
        print()

```

At first there are 2 functions used to initialize the game, initialize_game function includes the global variables used in the Game class and the initial state of the board.

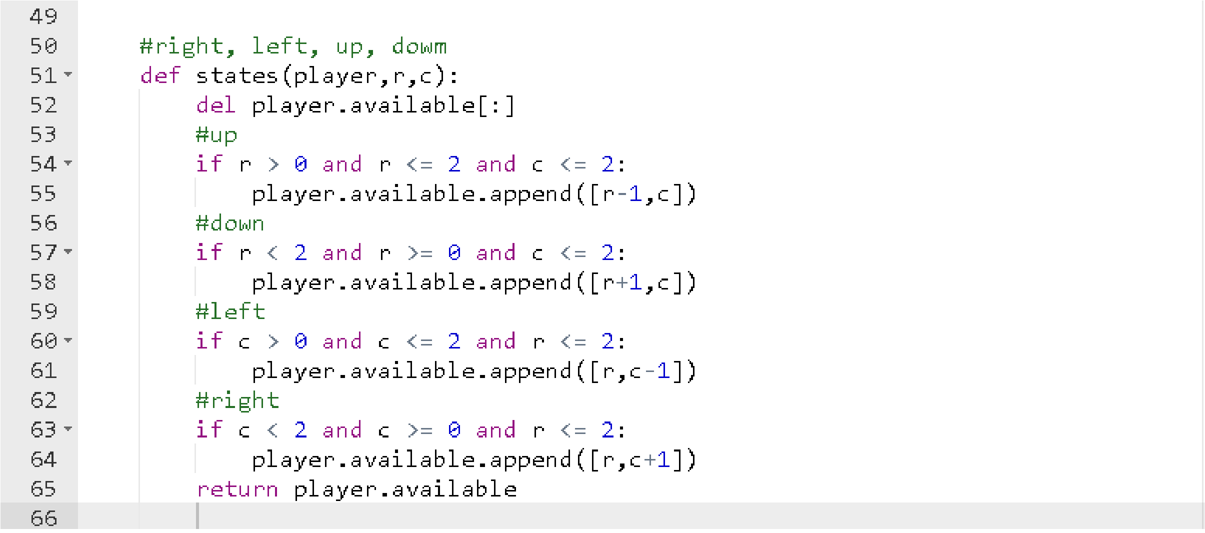
draw_board function is used to draw the board at each state.



```

    #Check if move is within the board range
    #Successor function
    def valid(player,r,c,x,y):
        res = player.states(x,y)
        if r < 0 or r > 2 or c < 0 or c > 2:
            return False
        #elif
        item = [r,c]
        if item in res:
            return True
        else:
            return False

    #Terminal state
    def end(player, moves,goal):
        if moves == 0:
            if player.score >= goal:
                return 'player'
            else:
                return 'fail'
        return None
```

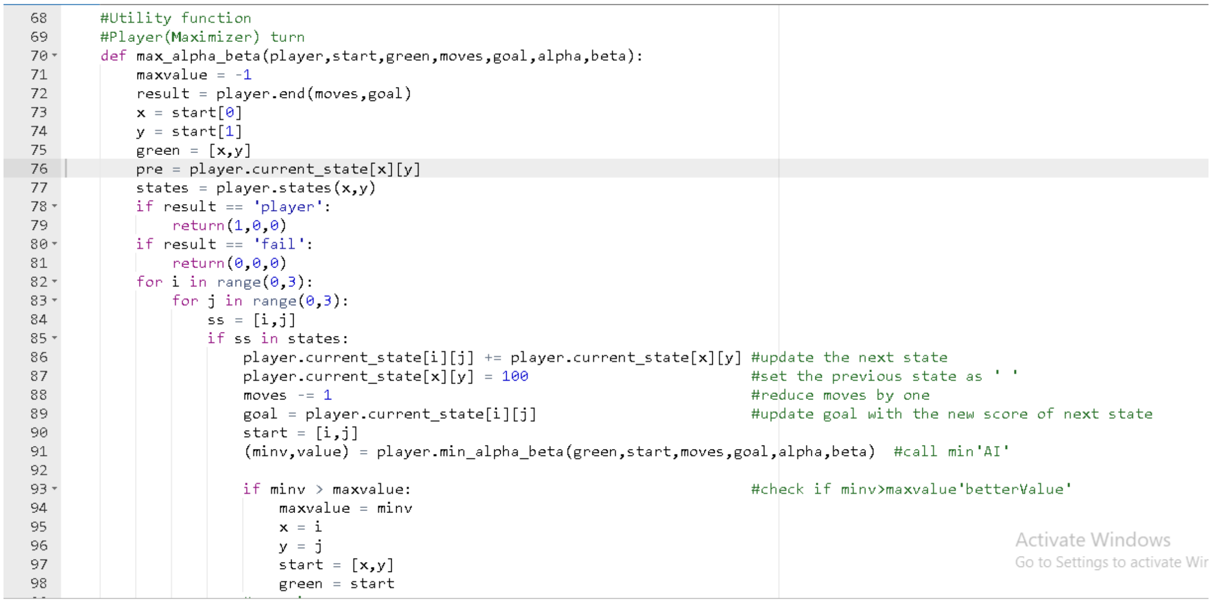
The successor function here is valid() which checks if the position is available to move to. It calls the states() function to get the available positions that the player can move through>

The terminal state: end() function which defines when the game stops by checking the number of moves and current score, if the moves = 0 it means there are no more moves available (reaches the maximum number of moves) then checks if the player score is equal to or greater than the minimum score entered before by the user. Other than that it returns None if there are more moves to use.



```

#right, left, up, dowm
def states(player,r,c):
    del player.available[:]
    #up
    if r > 0 and r <= 2 and c <= 2:
        player.available.append([r-1,c])
    #down
    if r < 2 and r >= 0 and c <= 2:
        player.available.append([r+1,c])
    #left
    if c > 0 and c <= 2 and r <= 2:
        player.available.append([r,c-1])
    #right
    if c < 2 and c >= 0 and r <= 2:
        player.available.append([r,c+1])
    return player.available

```

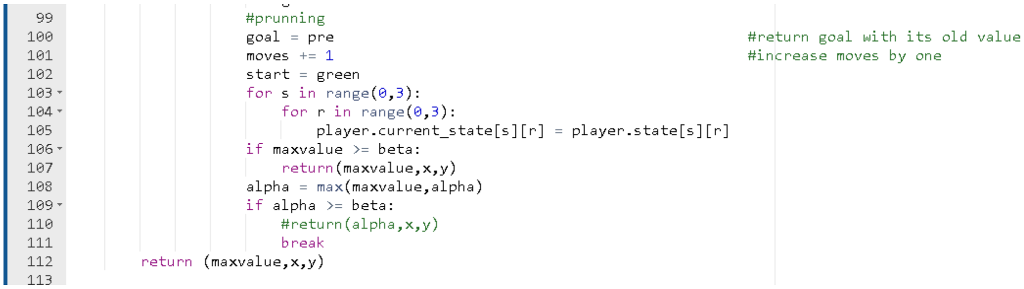
This function is to get the available moves that the player can move to (up, down, right, left).



```
#Utility function
#Player(Maximizer) turn
def max_alpha_beta(player,start,green,moves,goal,alpha,beta):
    maxvalue = -1
    result = player.end(moves,goal)
    x = start[0]
    y = start[1]
    green = [x,y]
    pre = player.current_state[x][y]
    states = player.states(x,y)
    if result == 'player':
        return(1,0,0)
    if result == 'fail':
        return(0,0,0)
    for i in range(0,3):
        for j in range(0,3):
            ss = [i,j]
            if ss in states:
                player.current_state[i][j] += player.current_state[x][y] #update the next state
                player.current_state[x][y] = 100                         #set the previous state as ' '
                moves -= 1                                               #reduce moves by one
                goal = player.current_state[i][j]                        #update goal with the new score of next state
                start = [i,j]
                (minv,value) = player.min_alpha_beta(green,start,moves,goal,alpha,beta)  #call min'AI'

                if minv > maxvalue:                                      #check if minv>maxvalue'betterValue'
                    maxvalue = minv
                    x = i
                    y = j
                    start = [x,y]
                    green = start
```

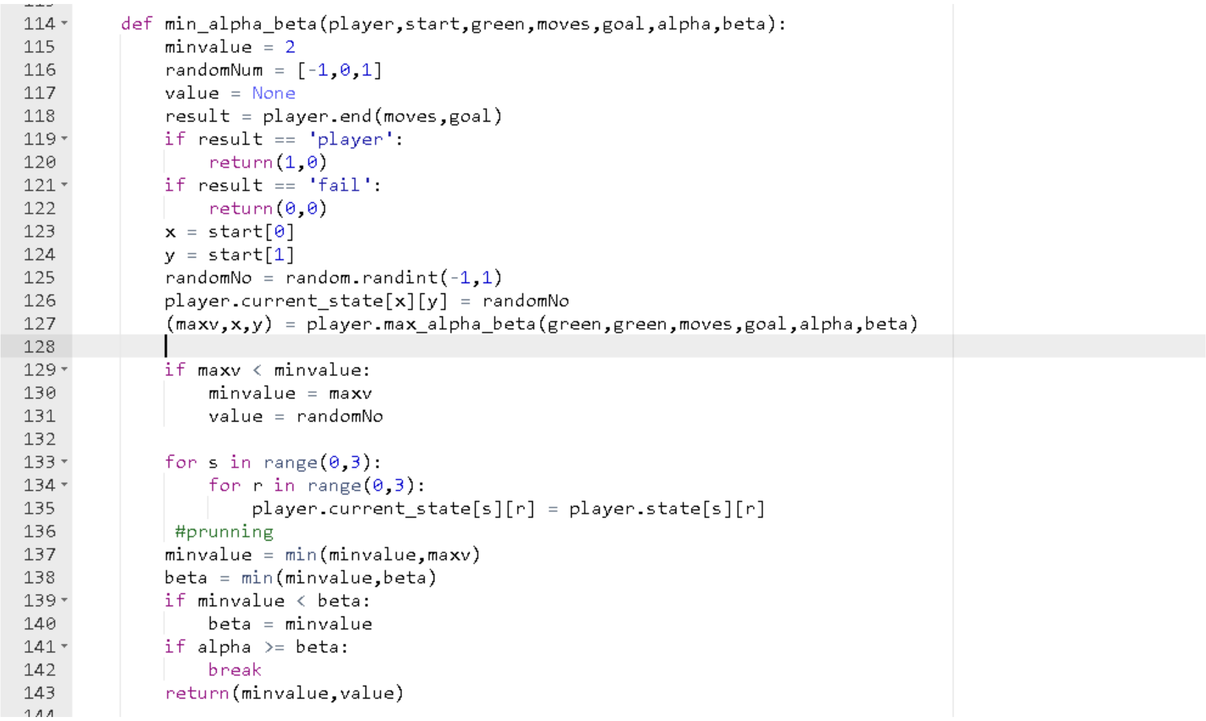
The max_alpha_beta() function is the function used to maximize the score, the player turn(max). First of all it checks if the game has ended if then it returns 1 if the player wins, 0 otherwise. The maxvalue initialized with -1 which is worse than the worst case which is 0.

‘start’ is a list includes the green tile (current position) if the game has not ended yet, looping among the player.current_state (board) and add the current position value to the next position to move if available, (states includes the available moves), decrease moves by one and update the goal (score) with the value of new position and update the value of previous position to 100 (means it is empty position the min will update) then update the start with the new position (x,y). calling the min_alpha_beta() function (the AI turn) to update the value of the previous position and return the min value, check if the minv is greater than the maxvalue (best value) then the max updated with the minv and the x,y new positions becomes i,j.



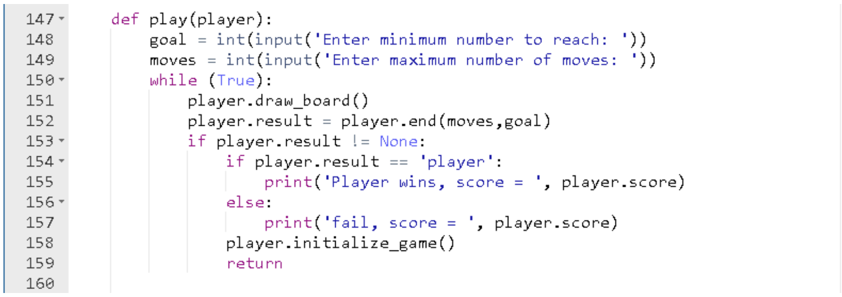
```
                #prunning
                goal = pre                                               #return goal with its old value
                moves += 1                                               #increase moves by one
                start = green
                for s in range(0,3):
                    for r in range(0,3):
                        player.current_state[s][r] = player.state[s][r]
                if maxvalue >= beta:
                    return(maxvalue,x,y)
                alpha = max(maxvalue,alpha)
                if alpha >= beta:
                    #return(alpha,x,y)
                    break
    return (maxvalue,x,y)
```

The rest of max_alpha_beta() function is to reset the values of variables (curretnt_state, moves, goal) to their previous values before applying the changes after each move. The last thing to apply is the pruning part, return the max between maxvalue and beta and max between maxvalue and alpha and update alpha with it, check if the alpha is greater than beta if true then break (ignore the rest nodes) and finally return the maxvalue, x and y (the best position indices to move.



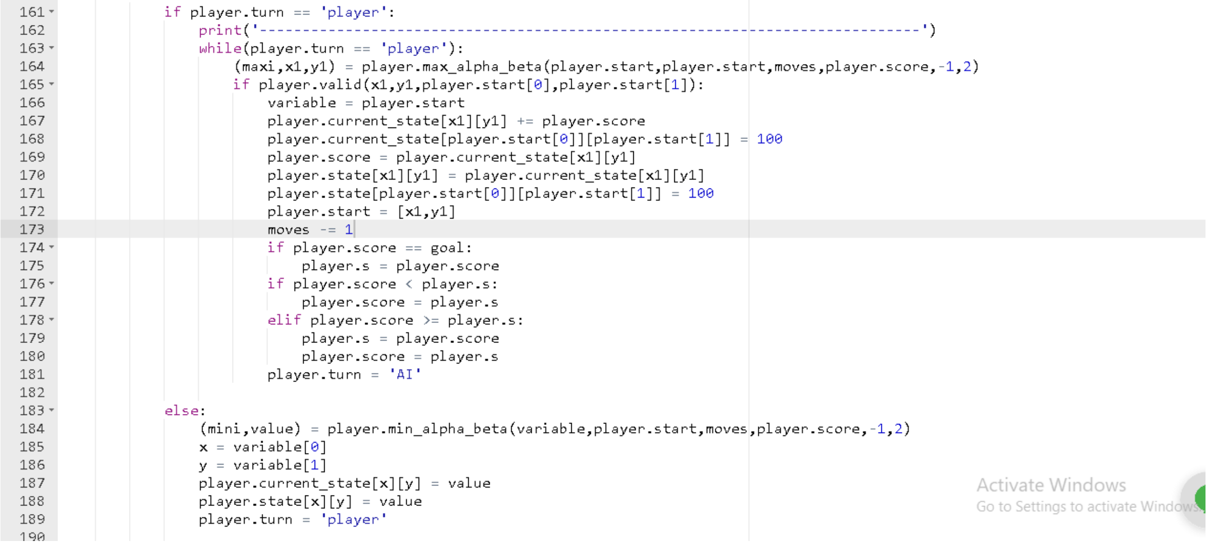
```
def min_alpha_beta(player,start,green,moves,goal,alpha,beta):
    minvalue = 2
    randomNum = [-1,0,1]
    value = None
    result = player.end(moves,goal)
    if result == 'player':
        return(1,0)
    if result == 'fail':
        return(0,0)
    x = start[0]
    y = start[1]
    randomNo = random.randint(-1,1)
    player.current_state[x][y] = randomNo
    (maxv,x,y) = player.max_alpha_beta(green,green,moves,goal,alpha,beta)

    if maxv < minvalue:
        minvalue = maxv
        value = randomNo

    for s in range(0,3):
        for r in range(0,3):
            player.current_state[s][r] = player.state[s][r]
     #prunning
    minvalue = min(minvalue,maxv)
    beta = min(minvalue,beta)
    if minvalue < beta:
        beta = minvalue
    if alpha >= beta:
        break
    return(minvalue,value)
```

The min_alpha_beta() function is the function used to minimize the score, the AI turn(min). First of all it checks if the game has ended if then it returns 1 if the player wins, 0 otherwise. The minvalue initialized with 2 which is better than the best case which is 1. ‘start’ is a list includes the green tile (current position) if the game has not ended yet, generate a random integer between [-1,1] and update the value of position to the number generated then call the max_alpha_beta() function (the player turn) to continue playing and return the max value, update the minvalue with the min value between minvalue and maxv then return the board to its previous state, update the beta with the min value between minvalue and beta then check if alpha is greater than beta break (ignore the rested nodes). Then return the minvalue and the value generated.



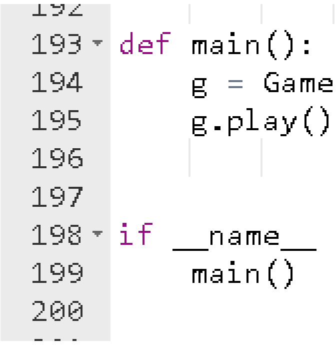
```python
def play(player):
    goal = int(input('Enter minimum number to reach: '))
    moves = int(input('Enter maximum number of moves: '))
    while (True):
        player.draw_board()
        player.result = player.end(moves,goal)
        if player.result != None:
            if player.result == 'player':
                print('Player wins, score = ', player.score)
            else:
                print('fail, score = ', player.score)
            player.initialize_game()
            return

```

The play(), takes the input from the user (goal and moves) then draws the current board state which is the initial at first, checks if the game has ended and prints if the player wins (with the score reached) or fails.



```python
        if player.turn == 'player':
            print('-----------------------------------------------------------------------------')
            while(player.turn == 'player'):
                (maxi,x1,y1) = player.max_alpha_beta(player.start,player.start,moves,player.score,-1,2)
                if player.valid(x1,y1,player.start[0],player.start[1]):
                    variable = player.start
                    player.current_state[x1][y1] += player.score
                    player.current_state[player.start[0]][player.start[1]] = 100
                    player.score = player.current_state[x1][y1]
                    player.state[x1][y1] = player.current_state[x1][y1]
                    player.state[player.start[0]][player.start[1]] = 100
                    player.start = [x1,y1]
                    moves -= 1
                    if player.score == goal:
                        player.s = player.score
                    if player.score < player.s:
                        player.score = player.s
                    elif player.score >= player.s:
                        player.s = player.score
                        player.score = player.s
                    player.turn = 'AI'

        else:
            (mini,value) = player.min_alpha_beta(variable,player.start,moves,player.score,-1,2)
            x = variable[0]
            y = variable[1]
            player.current_state[x][y] = value
            player.state[x][y] = value
            player.turn = 'player'

```

If the game has not ended, check the turn to play if it is the player turn then call the max_alpha_beta() which takes the player.start(position), moves, player.score, -1 as alpha and 2 as beta. Update the value of position returned by adding the player.score to it then update the previous position with 100 (the empty), update the current score with the new green value , decrease the moves by one and change the player.turn to ‘AI’ to play. There are some conditions used to save the score of the player if it reaches the goal and assign it to the score again if the player.score decreases while moving after reaching the goal. Else if it is the AI turn then call the min_alpha_beta() which returns the value to assign to the previous position (the player moved from) then change the player turn to ‘player’. Variable is a list contains the indices of the previous position.

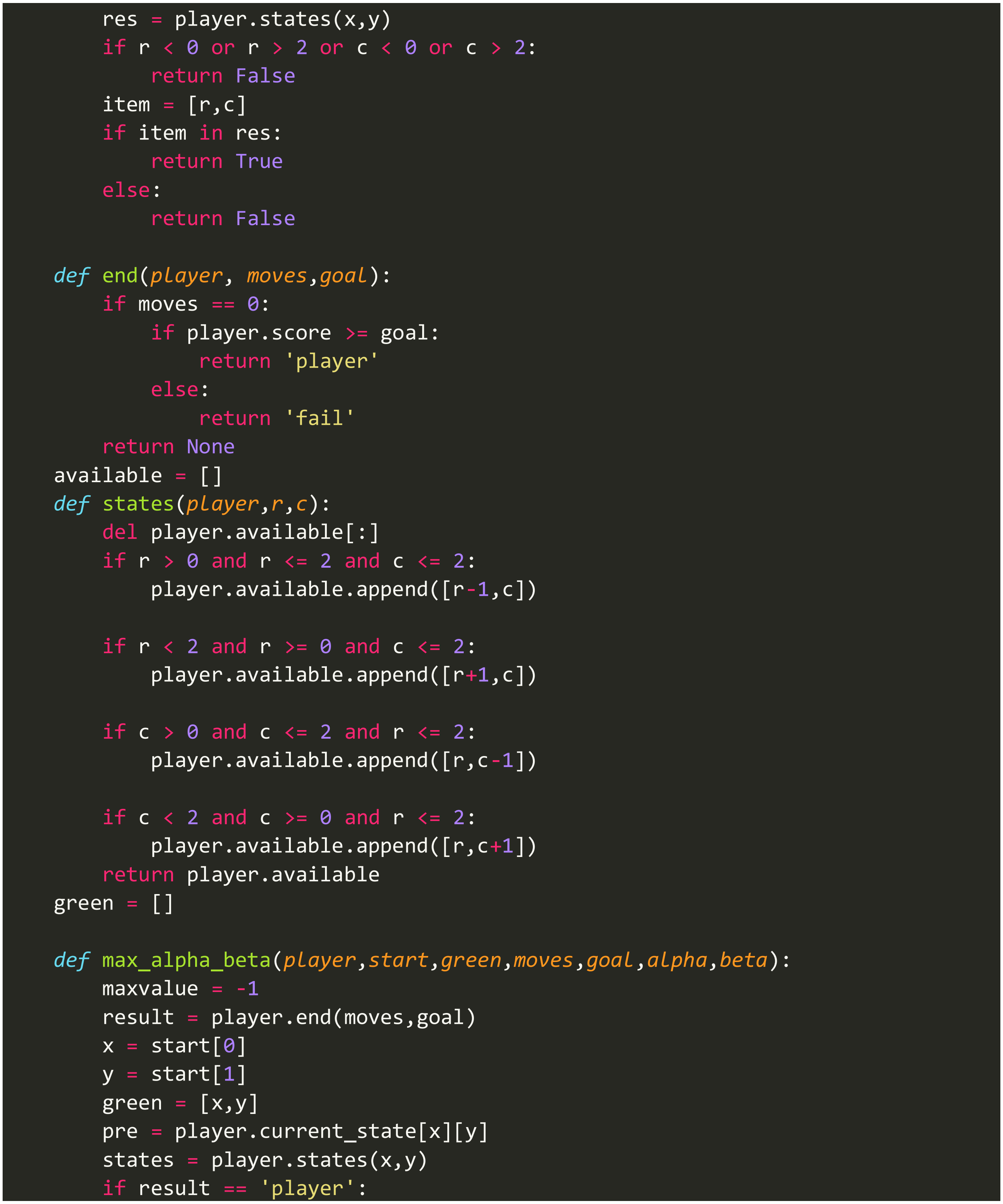


```
def main():
    g = Game()
    g.play()


if __name__ == "__main__":
    main()

```

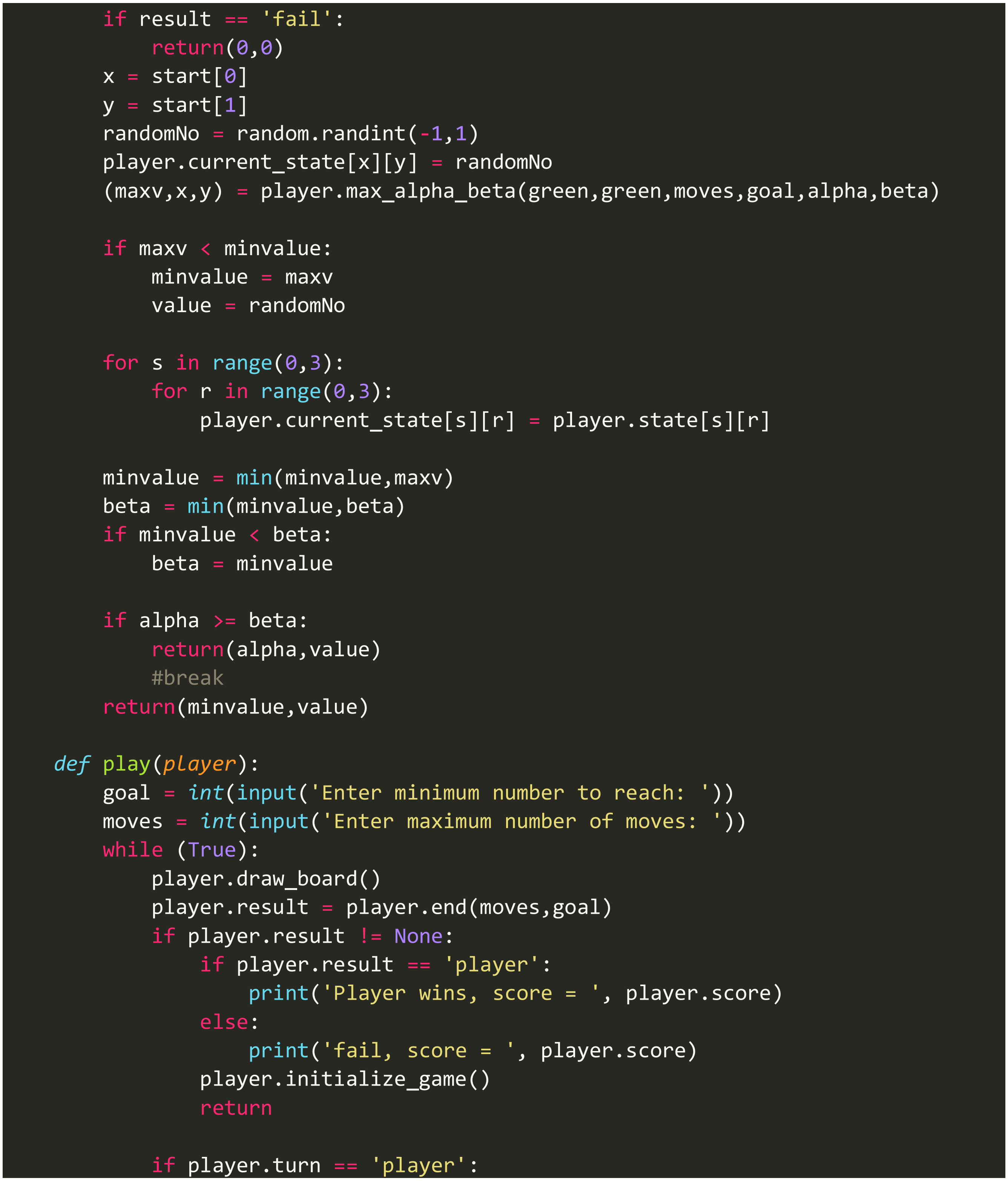
The main()-> create an object from Game() ‘g’, then call the play() function to start the game.

II. Code Listing:

```
import random
class Game:
    def __init__(player):
        player.initialize_game()

    def initialize_game(player):
        player.current_state = [[1,-1,0],
                              [1,0,1],
                              [0,1,-1]]
        player.score = player.current_state[2][0]
        player.s = player.score
        player.start = [2,0]
        variable = player.start
        player.turn = 'player'
        player.state = [[1,-1,0],
                        [1,0,1],
                        [0,1,-1]]

    def draw_board(player):
        for i in range(0, 3):
            for j in range(0, 3):
                print('{}|'.format(player.current_state[i][j]), end=" ")
            print()
        print()

    def valid(player,r,c,x,y):
```



```
        res = player.states(x,y)
        if r < 0 or r > 2 or c < 0 or c > 2:
            return False
        item = [r,c]
        if item in res:
            return True
        else:
            return False

    def end(player, moves,goal):
        if moves == 0:
            if player.score >= goal:
                return 'player'
            else:
                return 'fail'
        return None
    available = []
    def states(player,r,c):
        del player.available[:]
        if r > 0 and r <= 2 and c <= 2:
            player.available.append([r-1,c])

        if r < 2 and r >= 0 and c <= 2:
            player.available.append([r+1,c])

        if c > 0 and c <= 2 and r <= 2:
            player.available.append([r,c-1])

        if c < 2 and c >= 0 and r <= 2:
            player.available.append([r,c+1])
        return player.available
    green = []

    def max_alpha_beta(player,start,green,moves,goal,alpha,beta):
        maxvalue = -1
        result = player.end(moves,goal)
        x = start[0]
        y = start[1]
        green = [x,y]
        pre = player.current_state[x][y]
        states = player.states(x,y)
        if result == 'player':
            return(1,0,0)
        if result == 'fail':
            return(0,0,0)
        for i in range(0,3):
            for j in range(0,3):
                ss = [i,j]
                if ss in states:
                    player.current_state[i][j] += player.current_state[x][y]
                    player.current_state[x][y] = 100
                    moves -= 1
                    goal = player.current_state[i][j]
                    start = [i,j]
                    (minv,value) = player.min_alpha_beta(green,start,moves,goal,a
lpha,beta)

                    if minv > maxvalue:
                        maxvalue = minv
                        x = i
                        y = j
                        start = [x,y]
                        green = start
                    #prunning
                    goal = pre
                    moves += 1
                    start = green
                    for s in range(0,3):
                        for r in range(0,3):
                            player.current_state[s][r] = player.state[s][r]
                    if maxvalue >= beta:
                        return(maxvalue,x,y)
                    alpha = max(maxvalue,alpha)
                    if alpha >= beta:
                        break
        return (maxvalue,x,y)

    def min_alpha_beta(player,start,green,moves,goal,alpha,beta):
        minvalue = 2
        randomNum = [-1,0,1]
        value = None
        result = player.end(moves,goal)
        if result == 'player':
            return(1,0)
```

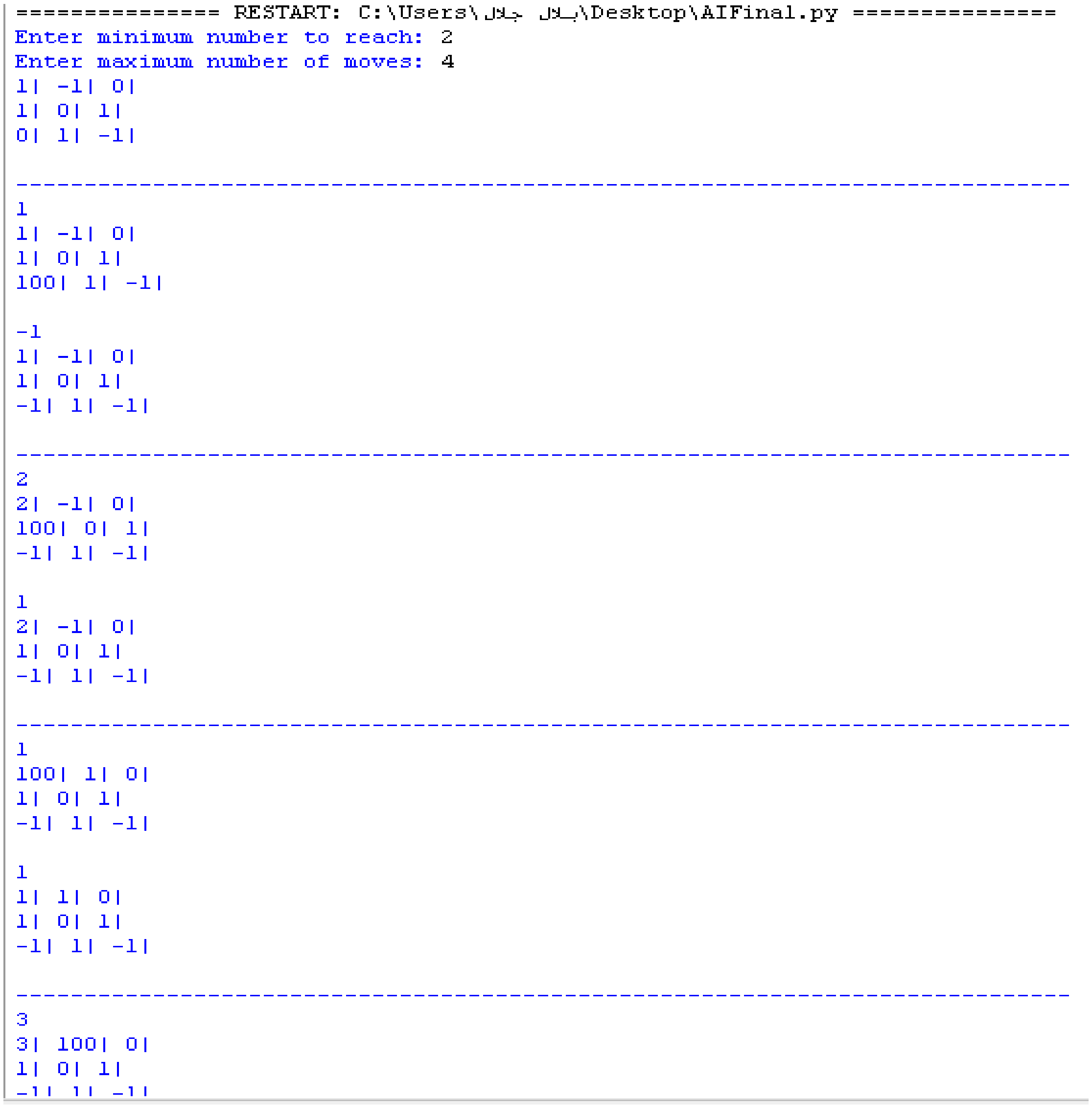


```
    if result == 'fail':
        return(0,0)
    x = start[0]
    y = start[1]
    randomNo = random.randint(-1,1)
    player.current_state[x][y] = randomNo
    (maxv,x,y) = player.max_alpha_beta(green,green,moves,goal,alpha,beta)

    if maxv < minvalue:
        minvalue = maxv
        value = randomNo

    for s in range(0,3):
        for r in range(0,3):
            player.current_state[s][r] = player.state[s][r]

    minvalue = min(minvalue,maxv)
    beta = min(minvalue,beta)
    if minvalue < beta:
        beta = minvalue

    if alpha >= beta:
        return(alpha,value)
        #break
    return(minvalue,value)

def play(player):
    goal = int(input('Enter minimum number to reach: '))
    moves = int(input('Enter maximum number of moves: '))
    while (True):
        player.draw_board()
        player.result = player.end(moves,goal)
        if player.result != None:
            if player.result == 'player':
                print('Player wins, score = ', player.score)
            else:
                print('fail, score = ', player.score)
            player.initialize_game()
            return

        if player.turn == 'player':
                print('-----------------------------------------------------------
-------------------')
                while(player.turn == 'player'):
                    (maxi,x1,y1) = player.max_alpha_beta(player.start,player.star
t,moves,player.score,-1,2)
                    if player.valid(x1,y1,player.start[0],player.start[1]):
                        variable = player.start
                        player.current_state[x1][y1] += player.score
                        player.current_state[player.start[0]][player.start[1]] =
100

                        player.score = player.current_state[x1][y1]
                        player.state[x1][y1] = player.current_state[x1][y1]
                        player.state[player.start[0]][player.start[1]] = 100
                        player.start = [x1,y1]
                        moves -= 1
                        if player.score == goal:
                            player.s = player.score
                        if player.score < player.s:
                            player.score = player.s
                        elif player.score >= player.s:
                            player.s = player.score
                            player.score = player.s
                        player.turn = 'AI'

            else:
                (mini,value) = player.min_alpha_beta(variable,player.start,moves,
player.score,-1,2)
                x = variable[0]
                y = variable[1]
                player.current_state[x][y] = value
                player.state[x][y] = value
                player.turn = 'player'



def main():
    g = Game()
    g.play()


if __name__ == "__main__":
    main()
```

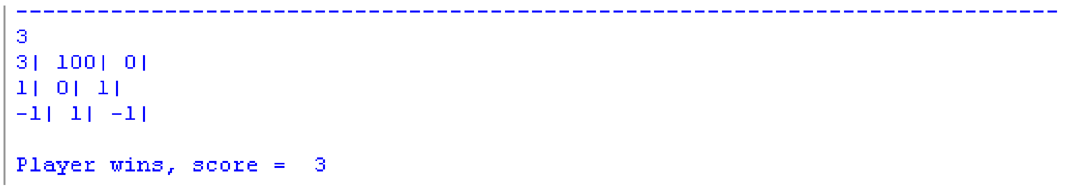
III. Test Cases:

Note: “100” number means the empty position that the min will update with a random number.

1-

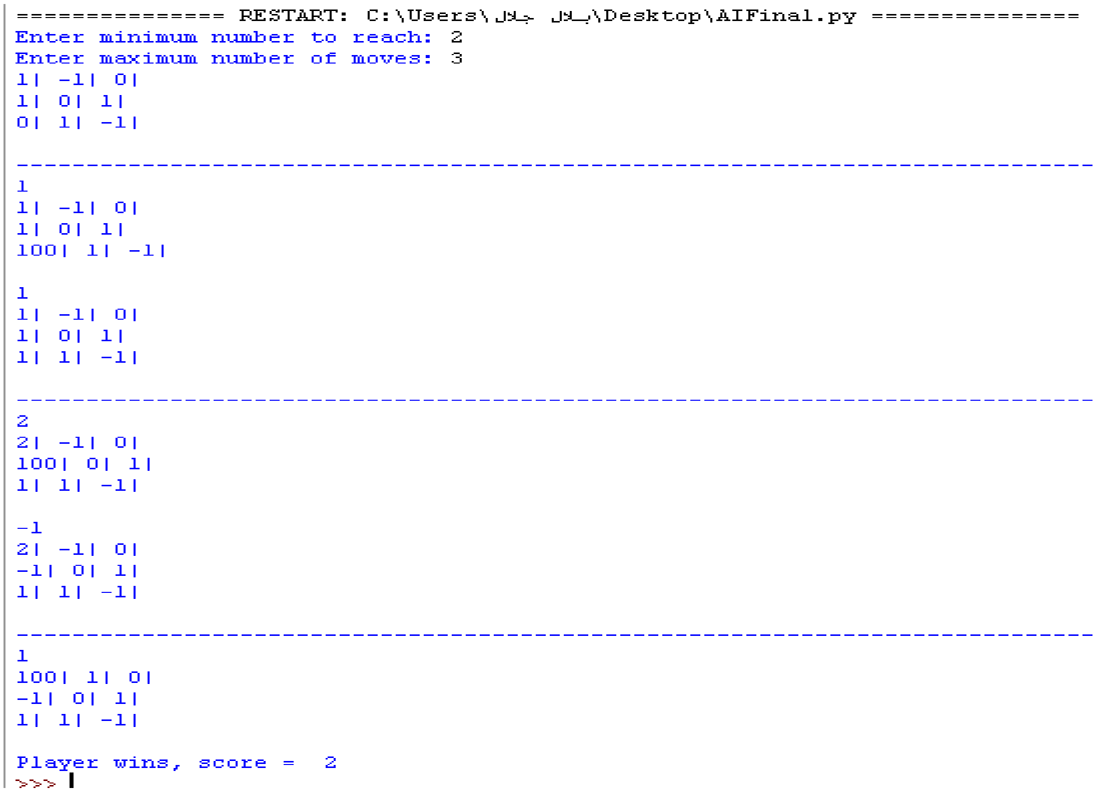


```
=============== RESTART: C:\Users\جلال بلال\Desktop\AIFinal.py ===============
Enter minimum number to reach: 2
Enter maximum number of moves: 4
1| -1| 0|
1| 0| 1|
0| 1| -1|

----------------------------------------------------------------------------
1
1| -1| 0|
1| 0| 1|
100| 1| -1|

-1
1| -1| 0|
1| 0| 1|
-1| 1| -1|

----------------------------------------------------------------------------
2
2| -1| 0|
100| 0| 1|
-1| 1| -1|

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

----------------------------------------------------------------------------
1
100| 1| 0|
1| 0| 1|
-1| 1| -1|

1
1| 1| 0|
1| 0| 1|
-1| 1| -1|

----------------------------------------------------------------------------
3
3| 100| 0|
1| 0| 1|
-1| 1| -1|
```



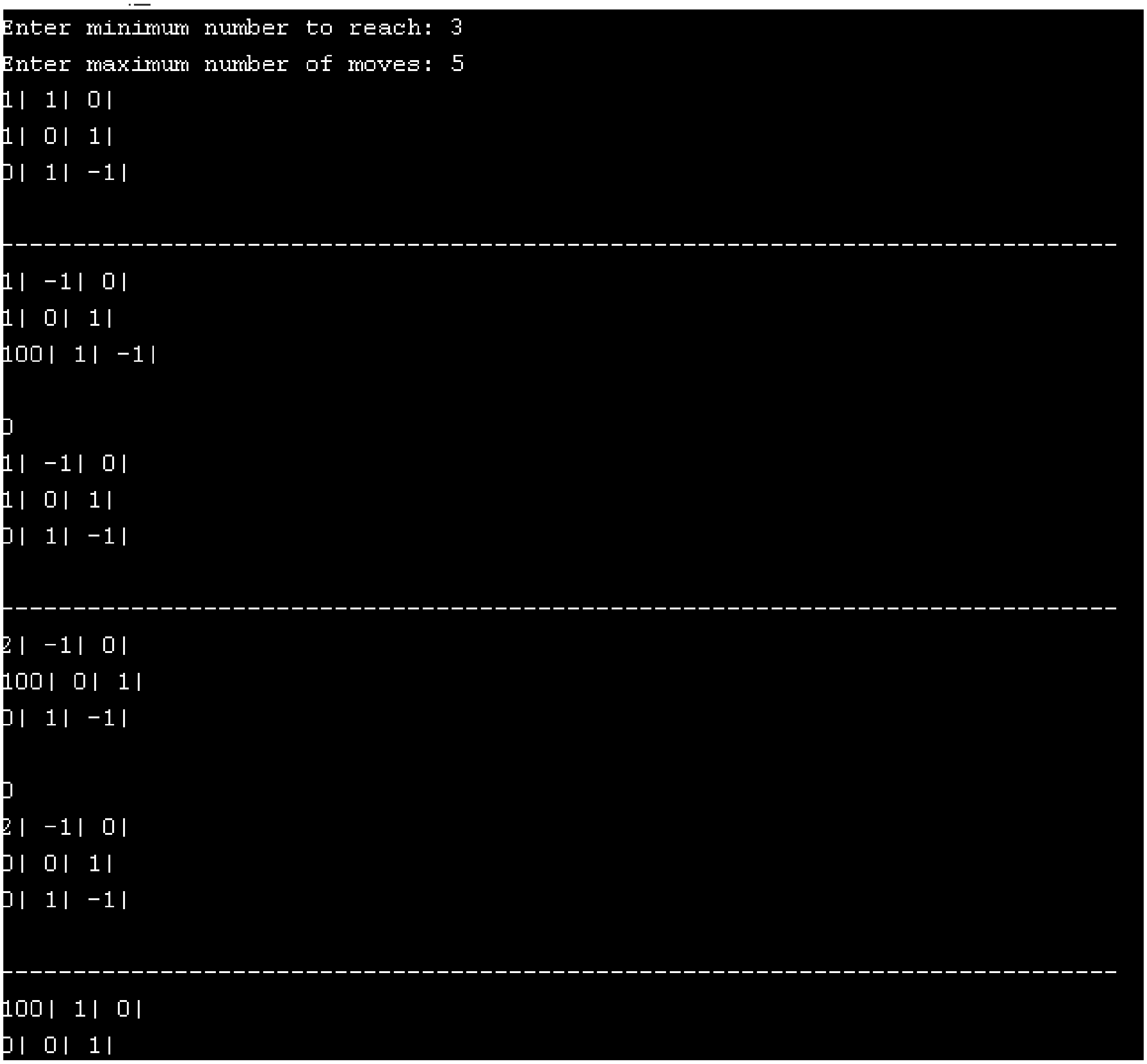
```
----------------------------------------------------------------------------
3
3| 100| 0|
1| 0| 1|
-1| 1| -1|

Player wins, score =  3
```

2-

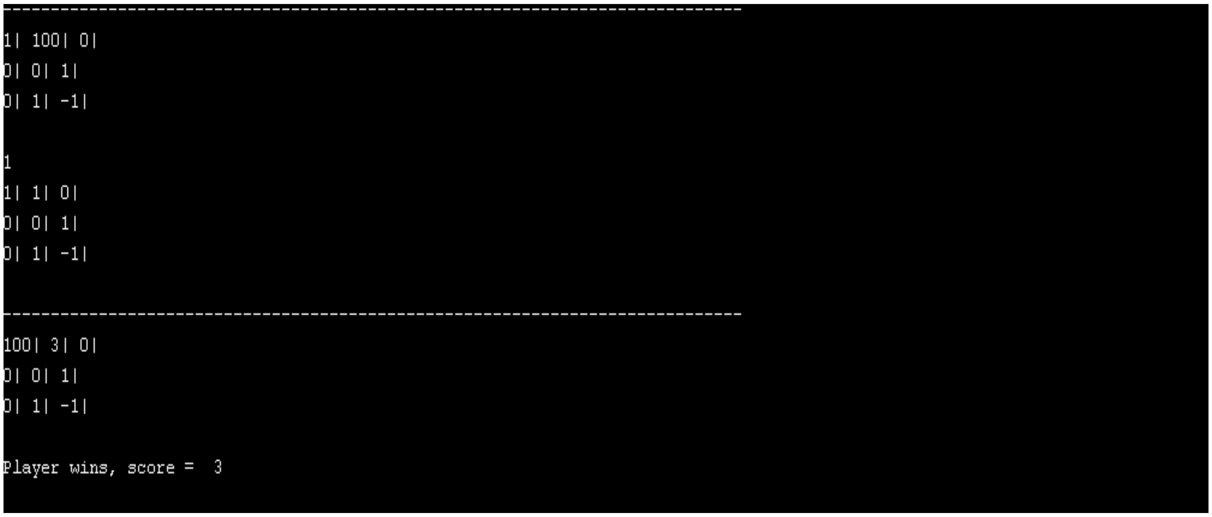


```
=============== RESTART: C:\Users\جلال بلال\Desktop\AIFinal.py ===============
Enter minimum number to reach: 2
Enter maximum number of moves: 3
1| -1| 0|
1| 0| 1|
0| 1| -1|

------------------------------------------------------------------------------
1
1| -1| 0|
1| 0| 1|
100| 1| -1|

1
1| -1| 0|
1| 0| 1|
1| 1| -1|

------------------------------------------------------------------------------
2
2| -1| 0|
100| 0| 1|
1| 1| -1|

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

------------------------------------------------------------------------------
1
100| 1| 0|
-1| 0| 1|
1| 1| -1|

Player wins, score =  2
>>> 
```

3-

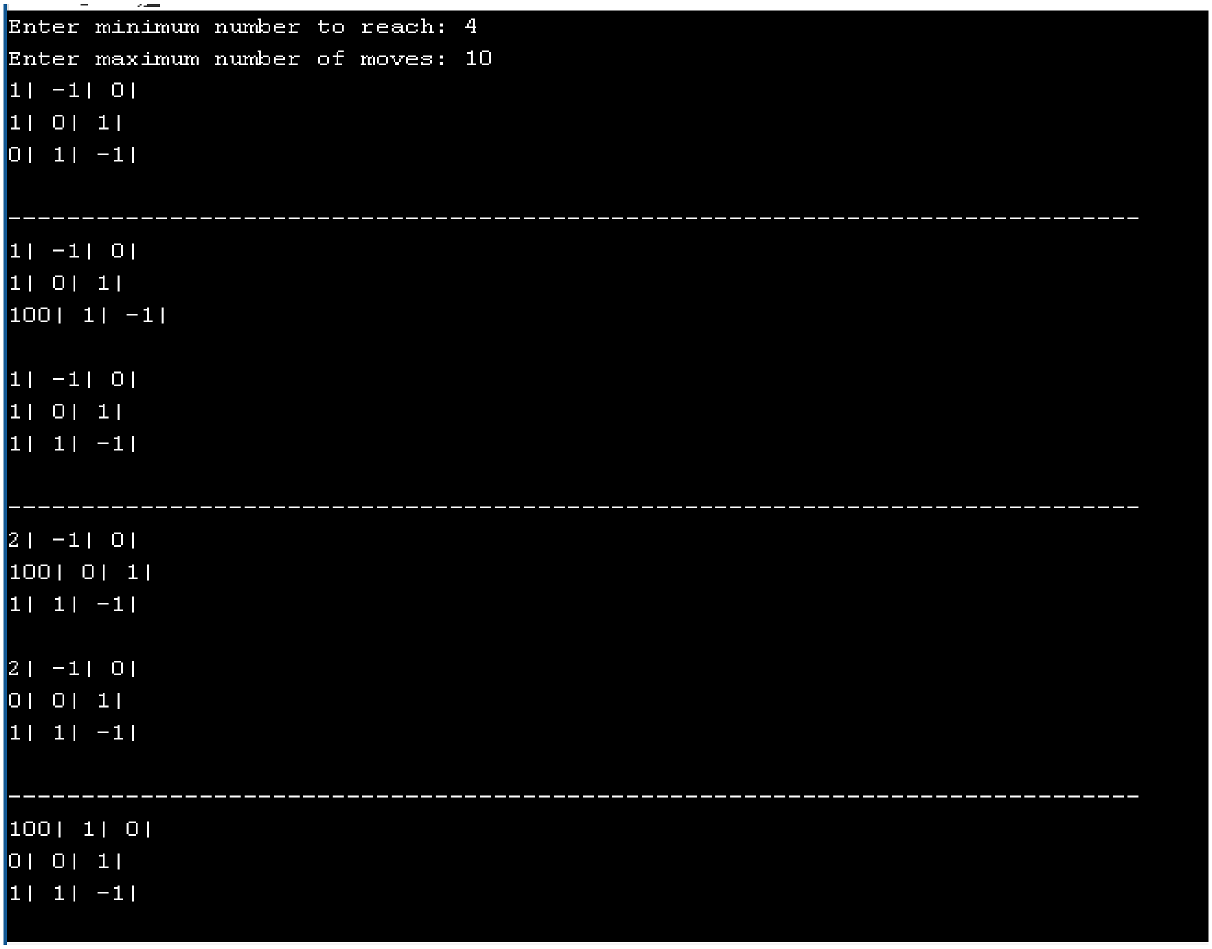


```
Enter minimum number to reach: 3
Enter maximum number of moves: 5
1| 1| 0|
1| 0| 1|
0| 1| -1|

------------------------------------------------------------------------------
1| -1| 0|
1| 0| 1|
100| 1| -1|

0
1| -1| 0|
1| 0| 1|
0| 1| -1|

------------------------------------------------------------------------------
2| -1| 0|
100| 0| 1|
0| 1| -1|

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

------------------------------------------------------------------------------
100| 1| 0|
0| 0| 1|
```

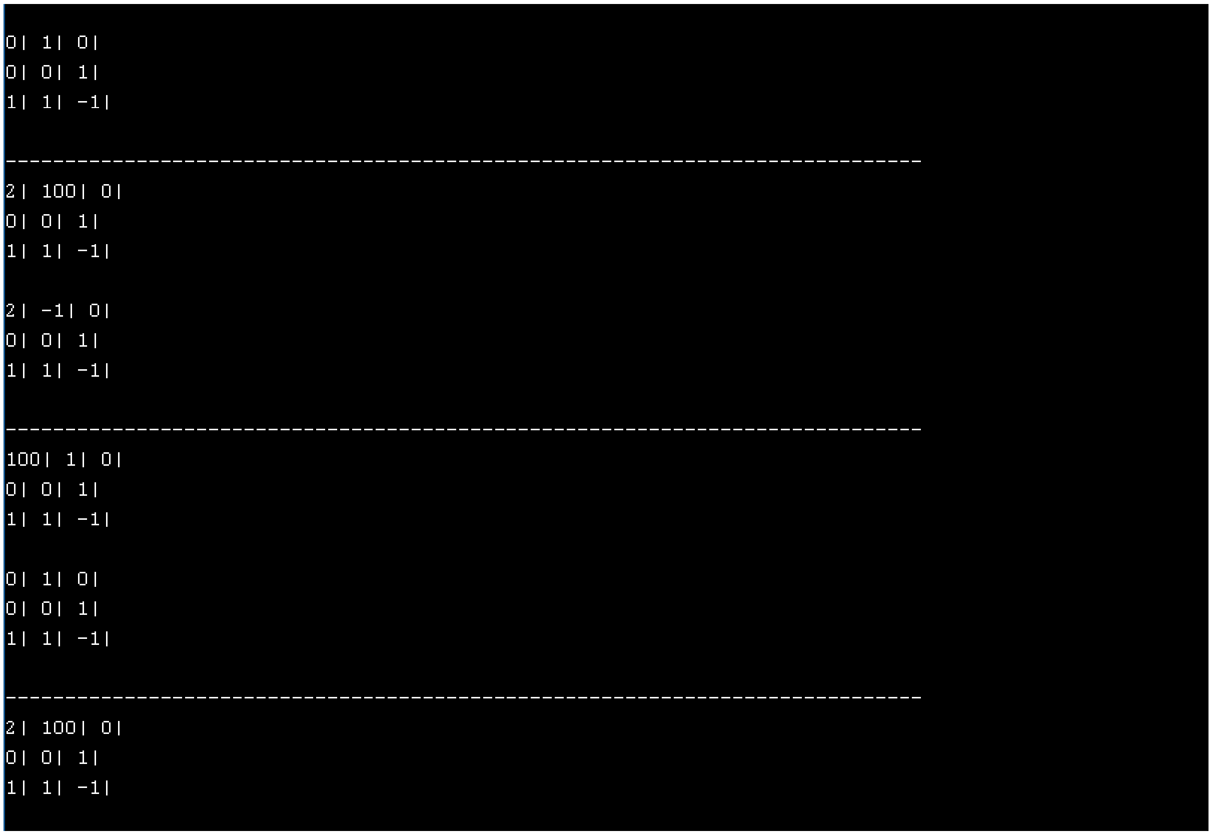


```
----------------------------------------------------------------------------
1| 100| 0|
0| 0| 1|
0| 1| -1|

1
1| 1| 0|
0| 0| 1|
0| 1| -1|


----------------------------------------------------------------------------
100| 3| 0|
0| 0| 1|
0| 1| -1|

Player wins, score =  3
```

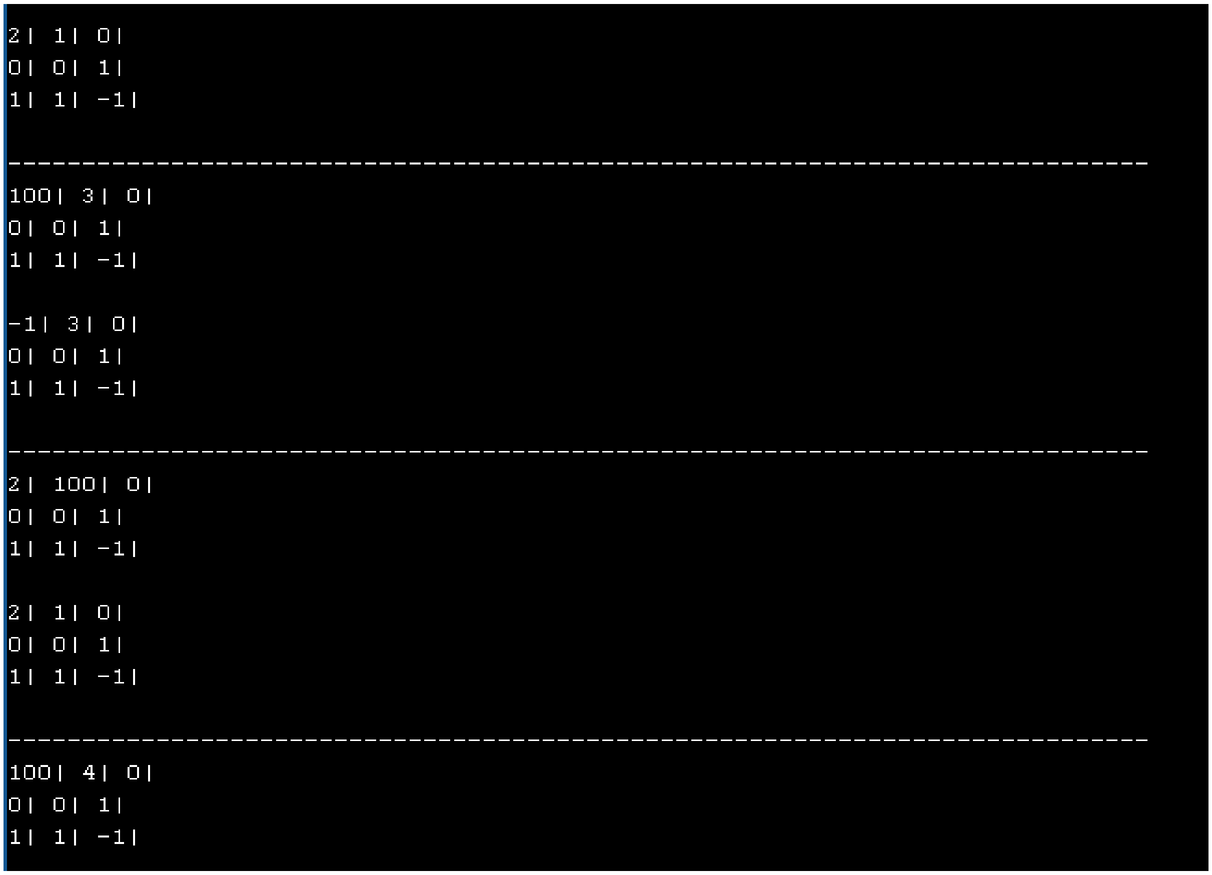
4-



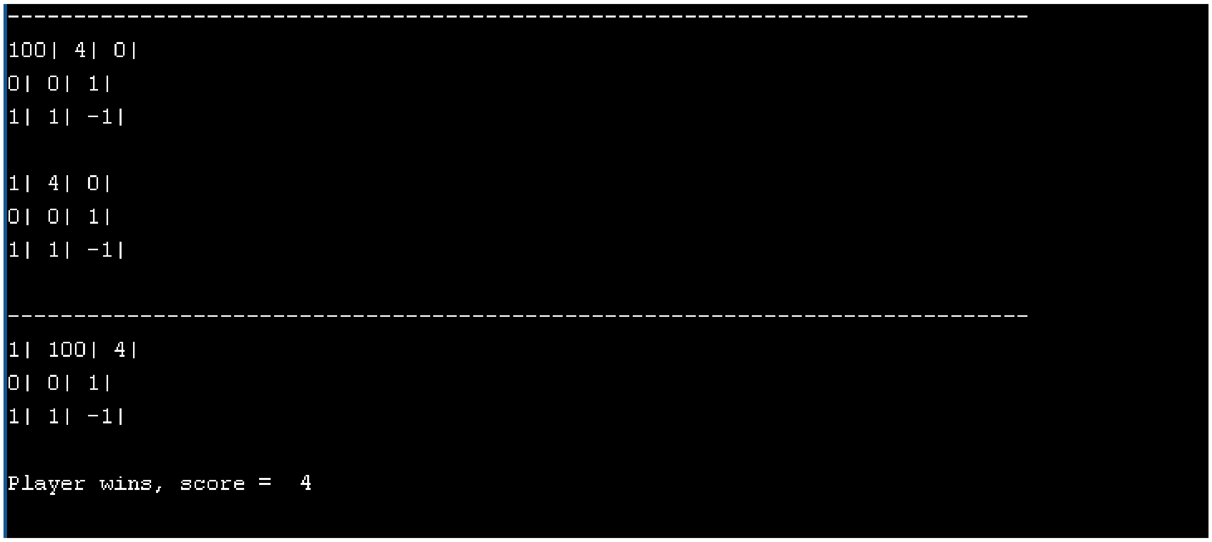
```
Enter minimum number to reach: 4
Enter maximum number of moves: 10
1| -1| 0|
1| 0| 1|
0| 1| -1|

----------------------------------------------------------------------------
1| -1| 0|
1| 0| 1|
100| 1| -1|

1| -1| 0|
1| 0| 1|
1| 1| -1|

----------------------------------------------------------------------------
2| -1| 0|
100| 0| 1|
1| 1| -1|

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

----------------------------------------------------------------------------
100| 1| 0|
0| 0| 1|
1| 1| -1|
```



```
0| 1| 0|
0| 0| 1|
1| 1| -1|

----------------------------------------------------------------------------
2| 100| 0|
0| 0| 1|
1| 1| -1|

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

----------------------------------------------------------------------------
100| 1| 0|
0| 0| 1|
1| 1| -1|

0| 1| 0|
0| 0| 1|
1| 1| -1|

----------------------------------------------------------------------------
2| 100| 0|
0| 0| 1|
1| 1| -1|
```

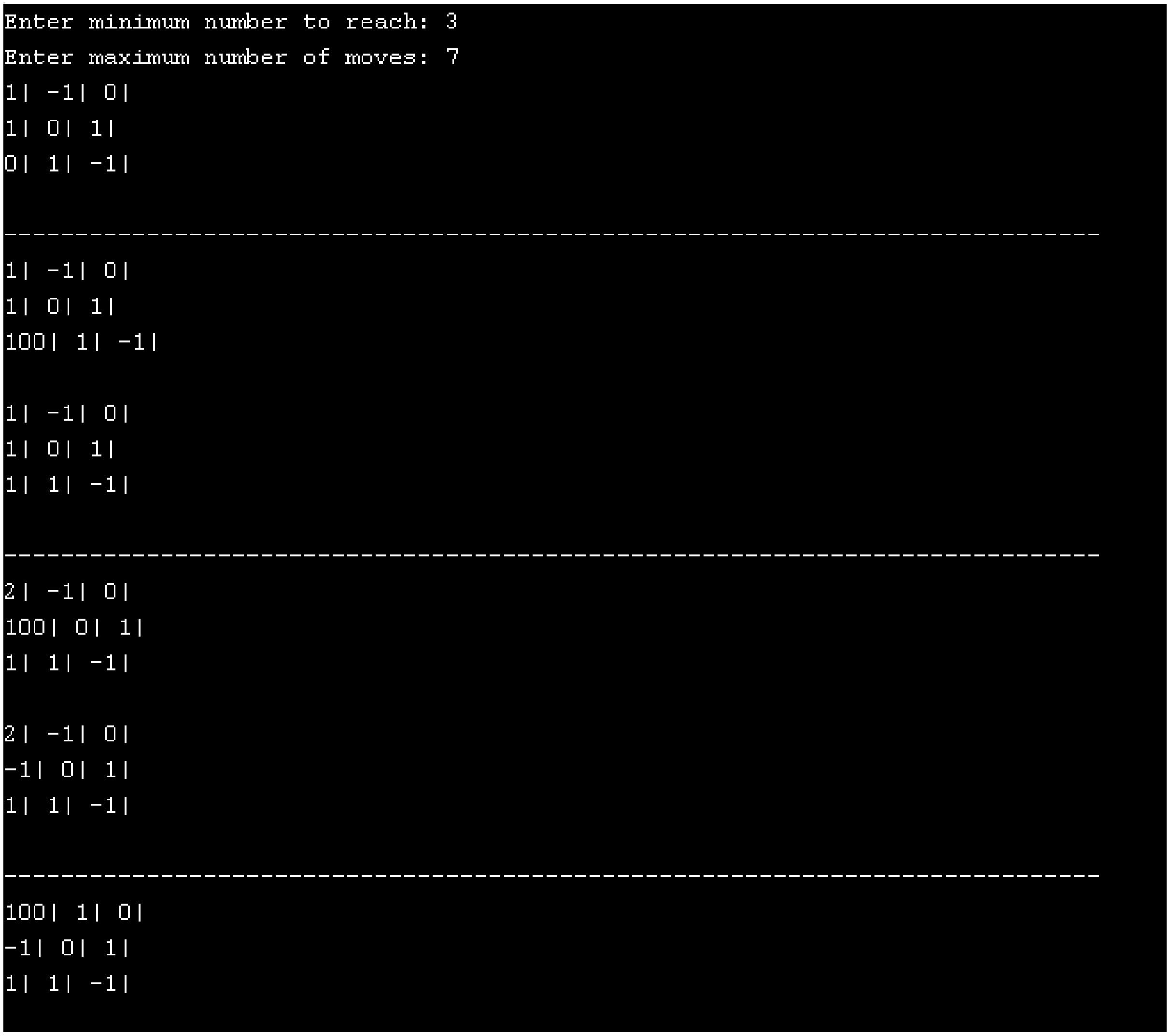


```
2| 1| 0|
0| 0| 1|
1| 1| -1|

-------------------------------------------------------------------------------
100| 3| 0|
0| 0| 1|
1| 1| -1|

-1| 3| 0|
0| 0| 1|
1| 1| -1|

-------------------------------------------------------------------------------
2| 100| 0|
0| 0| 1|
1| 1| -1|

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

-------------------------------------------------------------------------------
100| 4| 0|
0| 0| 1|
1| 1| -1|
```

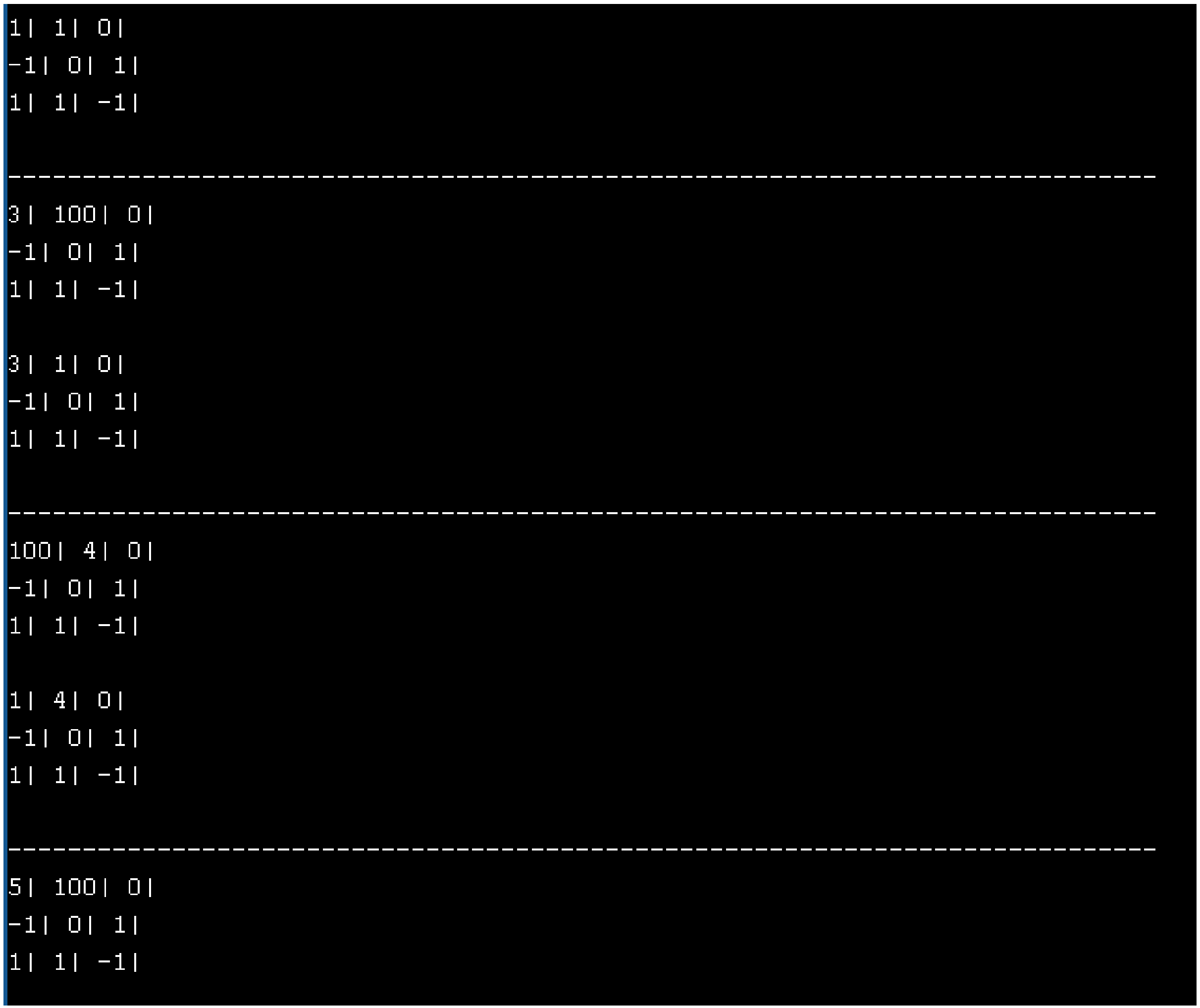


```
-------------------------------------------------------------------------------
100| 4| 0|
0| 0| 1|
1| 1| -1|

1| 4| 0|
0| 0| 1|
1| 1| -1|

-------------------------------------------------------------------------------
1| 100| 4|
0| 0| 1|
1| 1| -1|

Player wins, score =  4
```

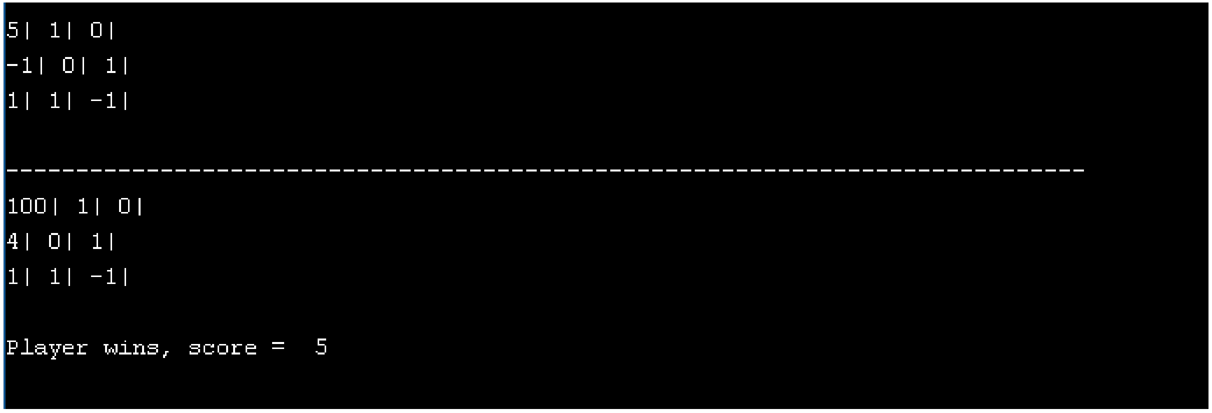
5-



```
Enter minimum number to reach: 3
Enter maximum number of moves: 7
1| -1| 0|
1| 0| 1|
0| 1| -1|

------------------------------------------------------------------------------
1| -1| 0|
1| 0| 1|
100| 1| -1|

1| -1| 0|
1| 0| 1|
1| 1| -1|

------------------------------------------------------------------------------
2| -1| 0|
100| 0| 1|
1| 1| -1|

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

------------------------------------------------------------------------------
100| 1| 0|
-1| 0| 1|
1| 1| -1|
```

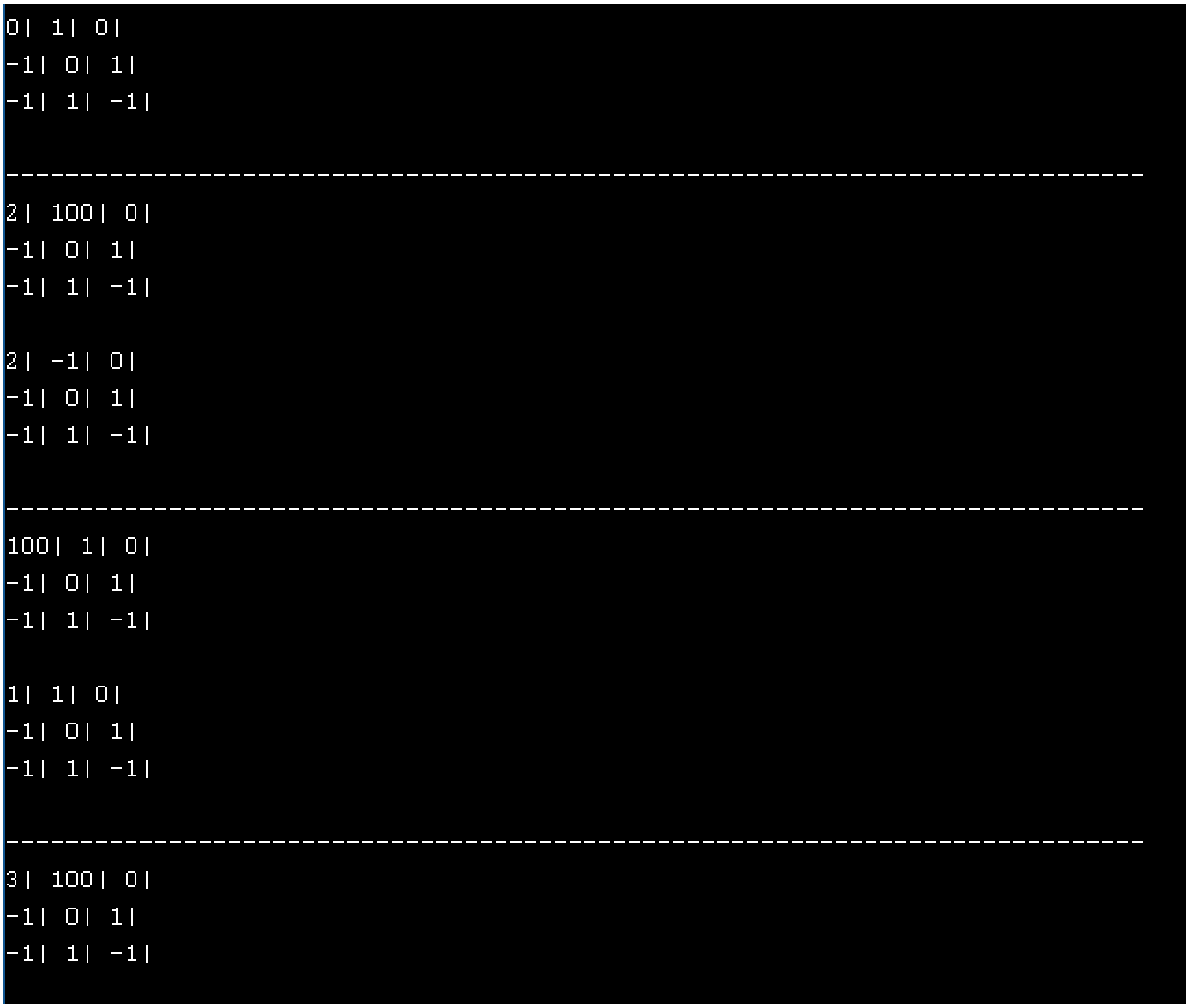


```
1| 1| 0|
-1| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------------
3| 100| 0|
-1| 0| 1|
1| 1| -1|

3| 1| 0|
-1| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------------
100| 4| 0|
-1| 0| 1|
1| 1| -1|

1| 4| 0|
-1| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------------
5| 100| 0|
-1| 0| 1|
1| 1| -1|
```

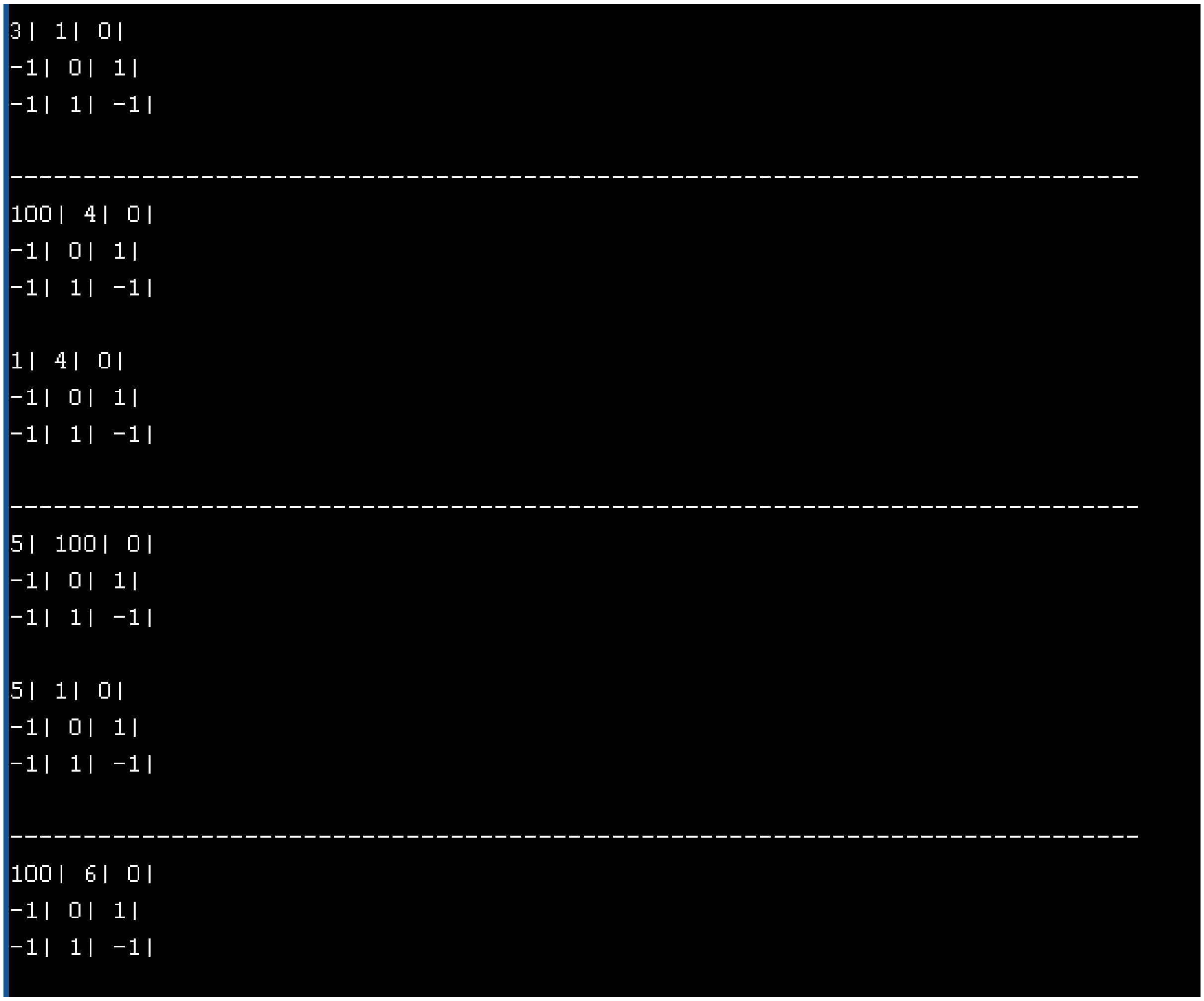


```
5| 1| 0|
-1| 0| 1|
1| 1| -1|

---------------------------------------------------------------------------
100| 1| 0|
4| 0| 1|
1| 1| -1|

Player wins, score =  5
```

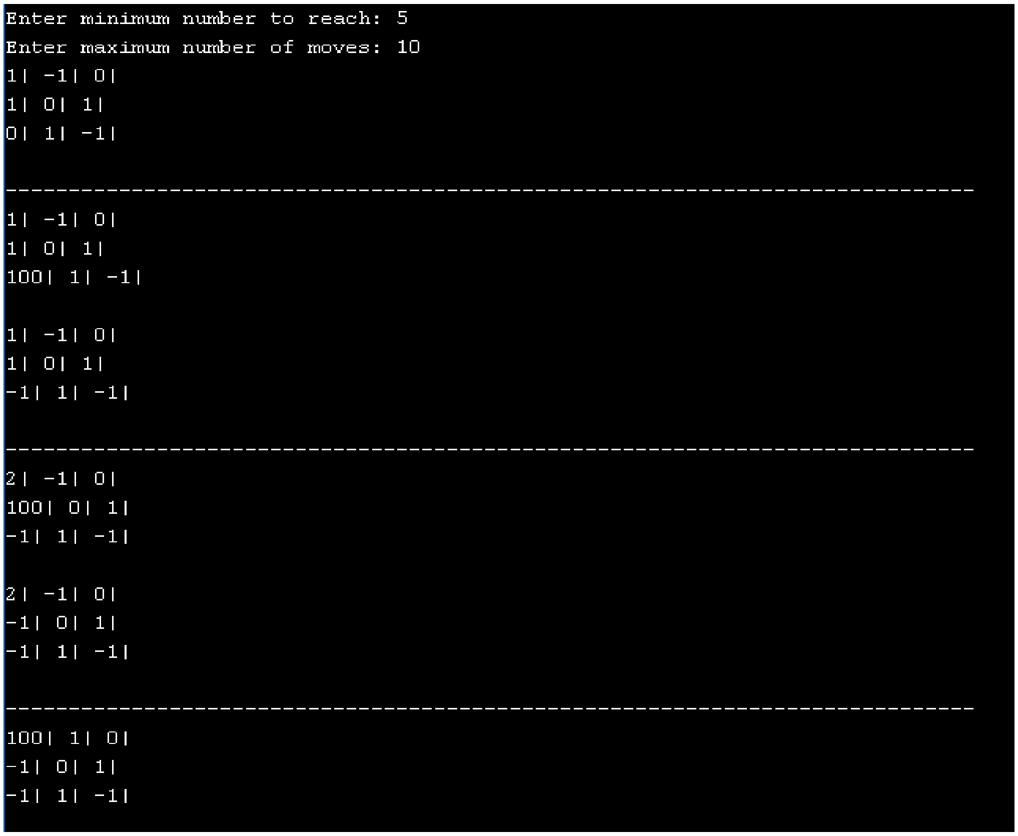
6-



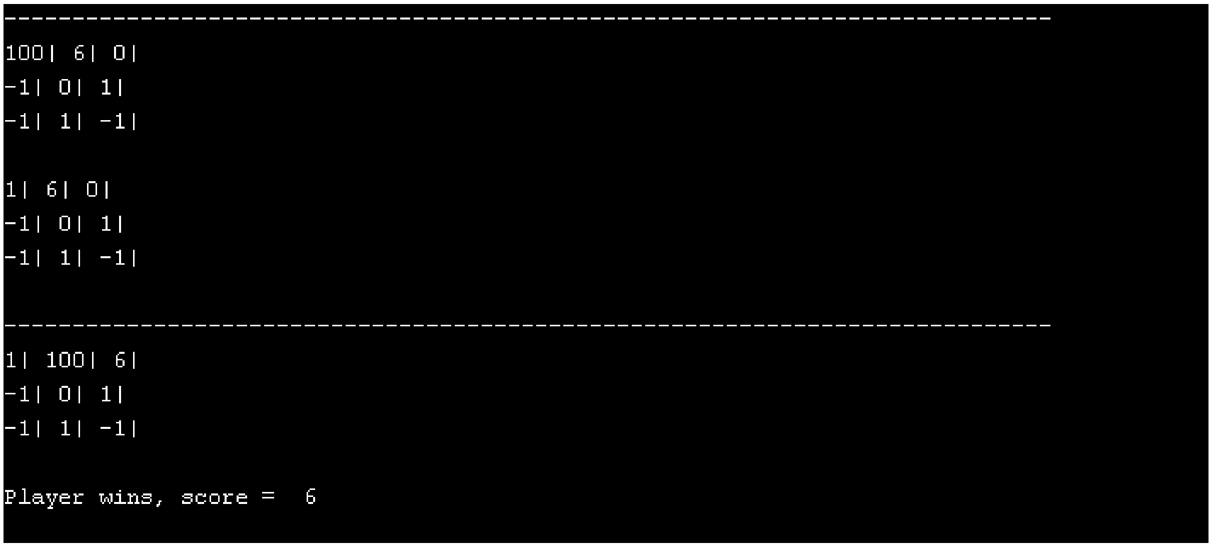
```
0| 1| 0|
-1| 0| 1|
-1| 1| -1|

-----------------------------------------------------------------------------
2| 100| 0|
-1| 0| 1|
-1| 1| -1|

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

-----------------------------------------------------------------------------
100| 1| 0|
-1| 0| 1|
-1| 1| -1|

1| 1| 0|
-1| 0| 1|
-1| 1| -1|

-----------------------------------------------------------------------------
3| 100| 0|
-1| 0| 1|
-1| 1| -1|
```



```
3| 1| 0|
-1| 0| 1|
-1| 1| -1|

------------------------------------------------------------------------------
100| 4| 0|
-1| 0| 1|
-1| 1| -1|

1| 4| 0|
-1| 0| 1|
-1| 1| -1|

------------------------------------------------------------------------------
5| 100| 0|
-1| 0| 1|
-1| 1| -1|

5| 1| 0|
-1| 0| 1|
-1| 1| -1|

------------------------------------------------------------------------------
100| 6| 0|
-1| 0| 1|
-1| 1| -1|
```

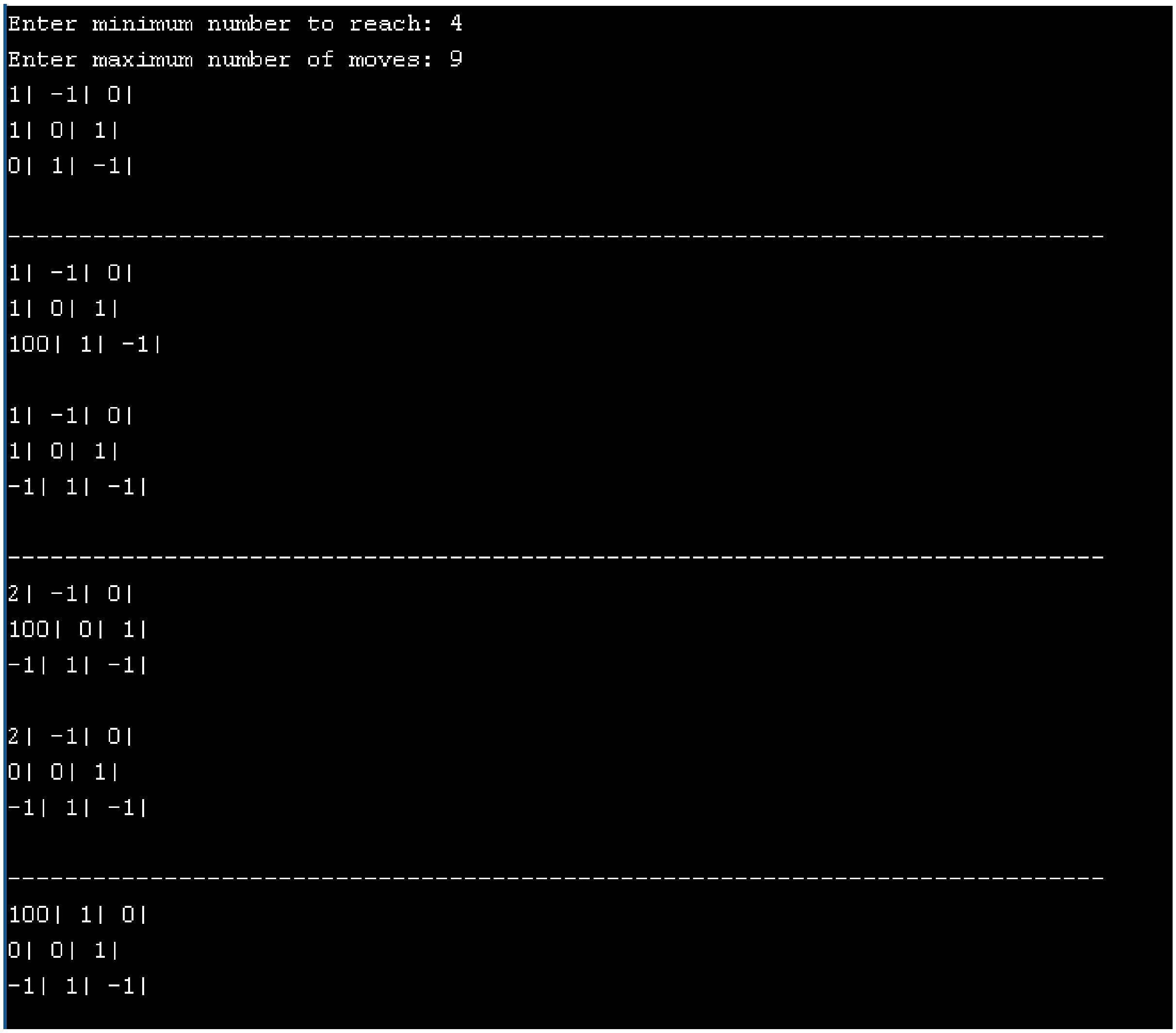


```
Enter minimum number to reach: 5
Enter maximum number of moves: 10
1| -1| 0|
1| 0| 1|
0| 1| -1|

---------------------------------------------------------------------------
1| -1| 0|
1| 0| 1|
100| 1| -1|

1| -1| 0|
1| 0| 1|
-1| 1| -1|

---------------------------------------------------------------------------
2| -1| 0|
100| 0| 1|
-1| 1| -1|

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

---------------------------------------------------------------------------
100| 1| 0|
-1| 0| 1|
-1| 1| -1|
```

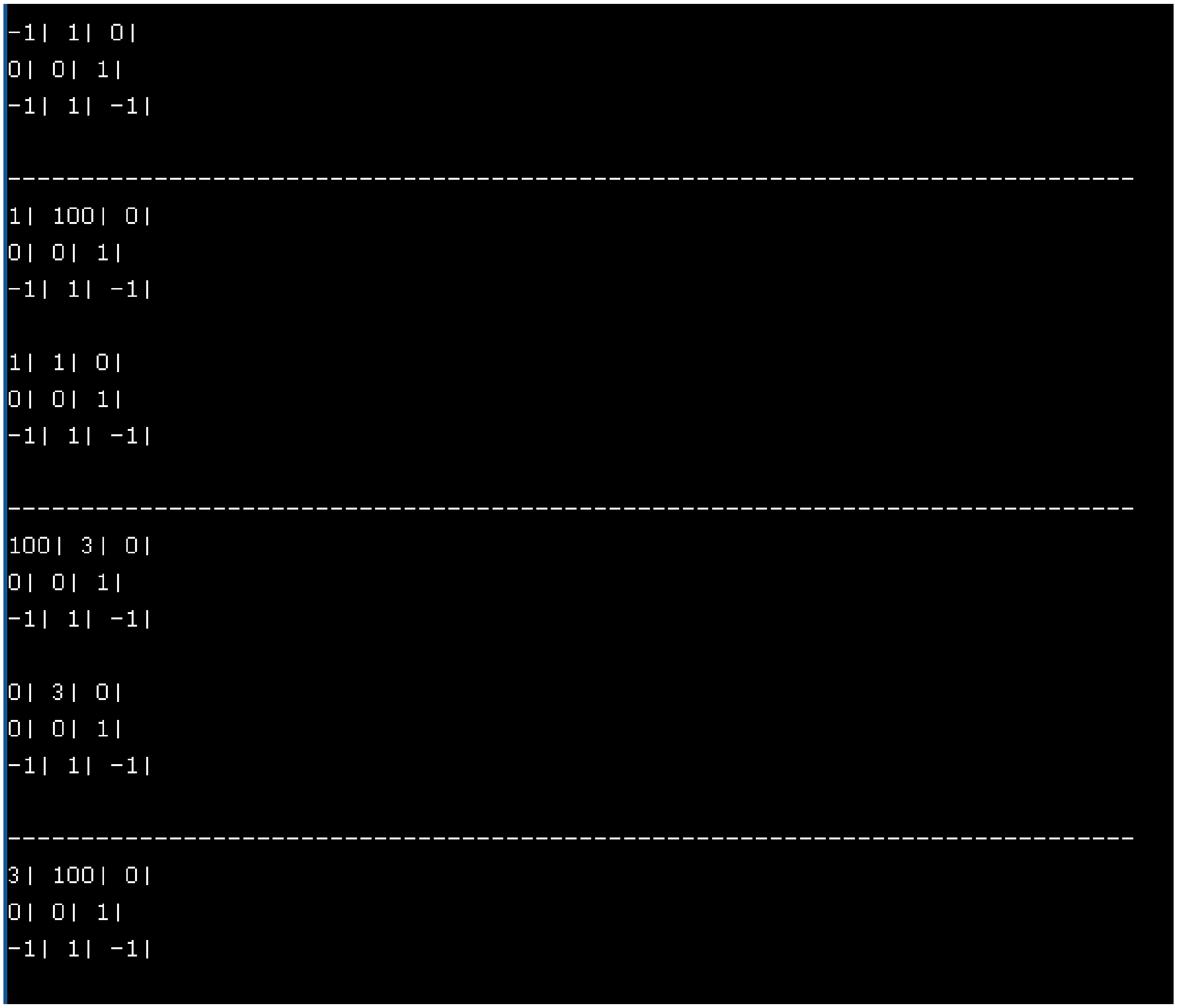


```
---------------------------------------------------------------------------
100| 6| 0|
-1| 0| 1|
-1| 1| -1|

1| 6| 0|
-1| 0| 1|
-1| 1| -1|

---------------------------------------------------------------------------
1| 100| 6|
-1| 0| 1|
-1| 1| -1|

Player wins, score =  6
```

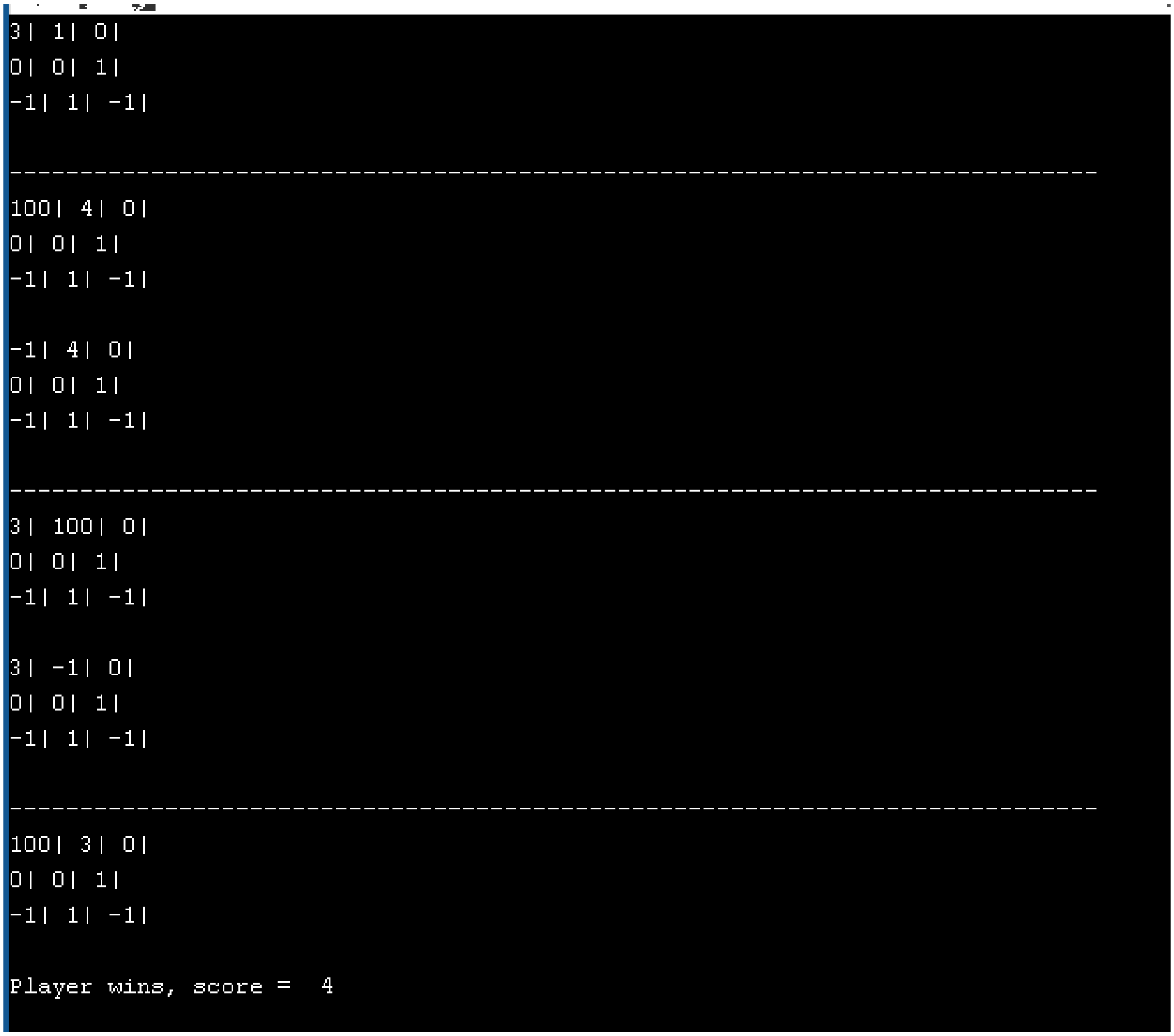
7-



```
Enter minimum number to reach: 4
Enter maximum number of moves: 9
1| -1| 0|
1| 0| 1|
0| 1| -1|

-------------------------------------------------------------------------------
1| -1| 0|
1| 0| 1|
100| 1| -1|

1| -1| 0|
1| 0| 1|
-1| 1| -1|

-------------------------------------------------------------------------------
2| -1| 0|
100| 0| 1|
-1| 1| -1|

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

-------------------------------------------------------------------------------
100| 1| 0|
0| 0| 1|
-1| 1| -1|
```

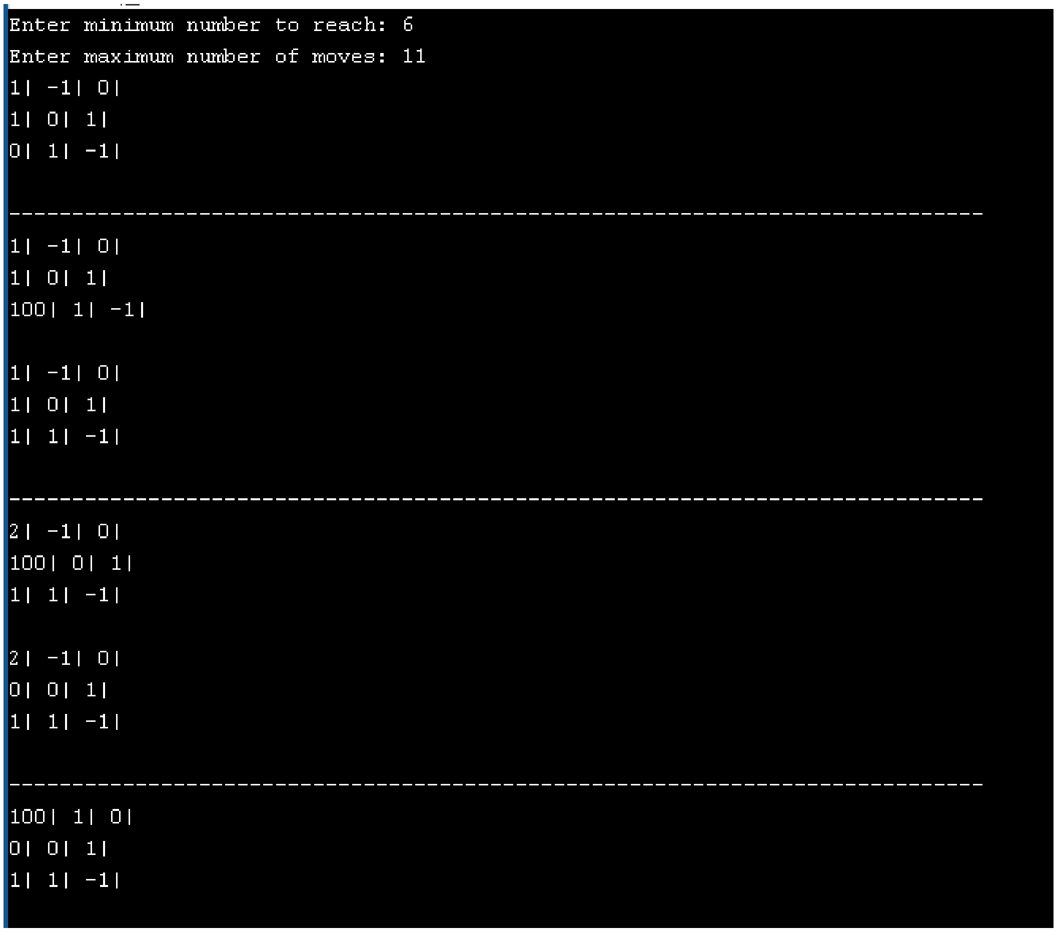


```
-1| 1| 0|
0| 0| 1|
-1| 1| -1|

-------------------------------------------------------------------------------
1| 100| 0|
0| 0| 1|
-1| 1| -1|

1| 1| 0|
0| 0| 1|
-1| 1| -1|

-------------------------------------------------------------------------------
100| 3| 0|
0| 0| 1|
-1| 1| -1|

0| 3| 0|
0| 0| 1|
-1| 1| -1|

-------------------------------------------------------------------------------
3| 100| 0|
0| 0| 1|
-1| 1| -1|
```

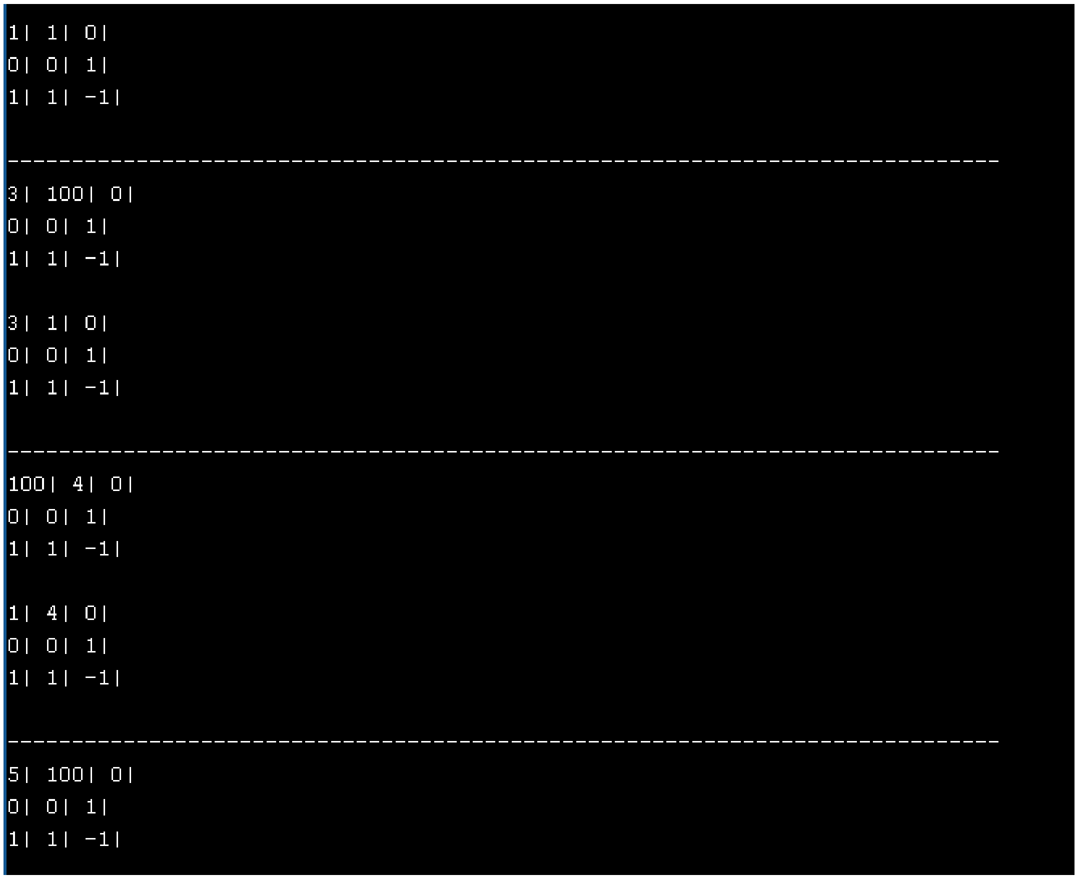


```
3| 1| 0|
0| 0| 1|
-1| 1| -1|

-----------------------------------------------------------------------------
100| 4| 0|
0| 0| 1|
-1| 1| -1|

-1| 4| 0|
0| 0| 1|
-1| 1| -1|

-----------------------------------------------------------------------------
3| 100| 0|
0| 0| 1|
-1| 1| -1|

3| -1| 0|
0| 0| 1|
-1| 1| -1|

-----------------------------------------------------------------------------
100| 3| 0|
0| 0| 1|
-1| 1| -1|

Player wins, score =  4
```

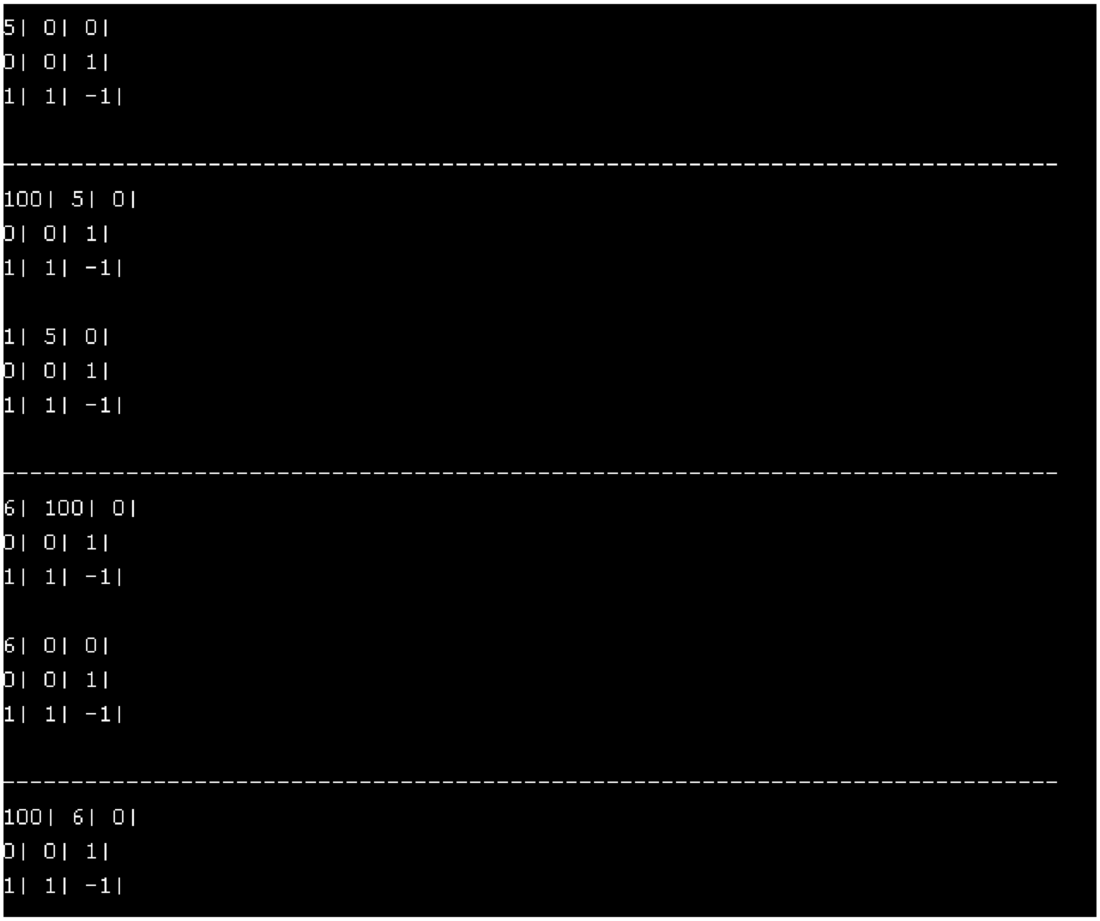
8-



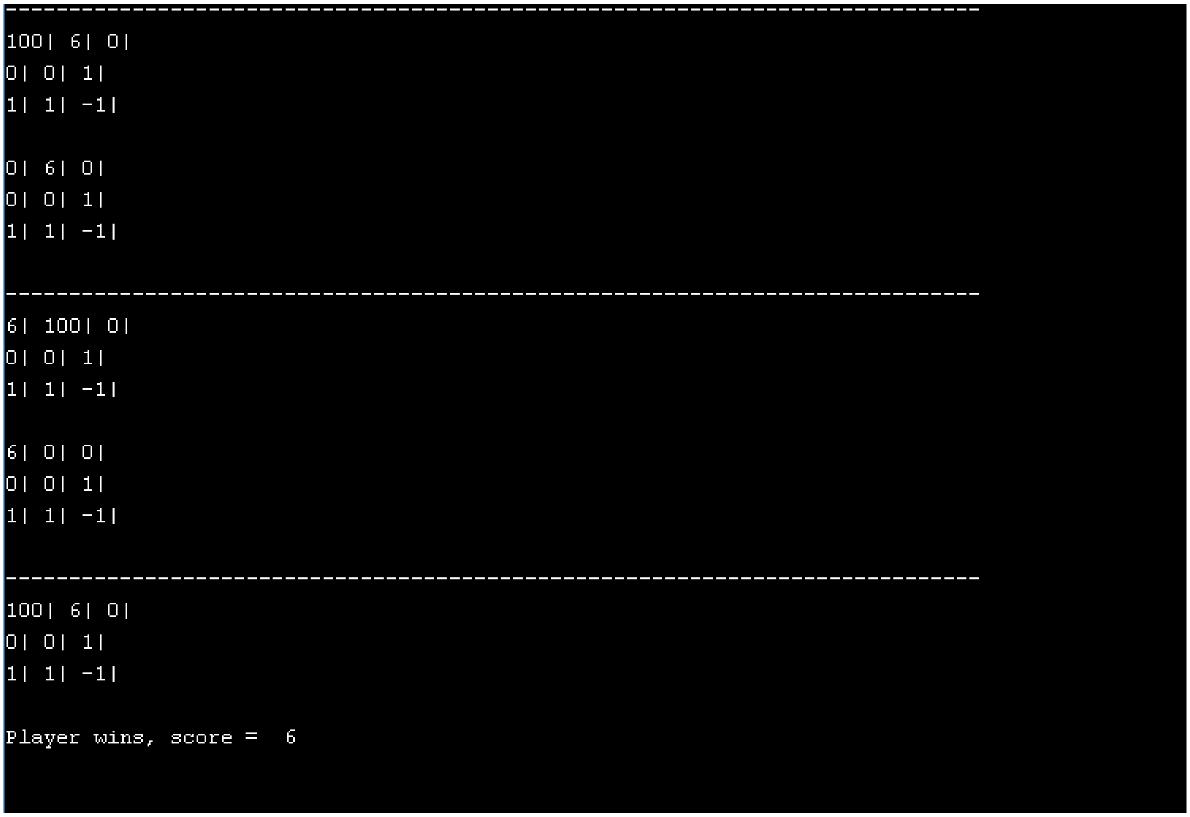
```
Enter minimum number to reach: 6
Enter maximum number of moves: 11
1| -1| 0|
1| 0| 1|
0| 1| -1|

-----------------------------------------------------------------------------
1| -1| 0|
1| 0| 1|
100| 1| -1|

1| -1| 0|
1| 0| 1|
1| 1| -1|

-----------------------------------------------------------------------------
2| -1| 0|
100| 0| 1|
1| 1| -1|

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

-----------------------------------------------------------------------------
100| 1| 0|
0| 0| 1|
1| 1| -1|
```



```
1| 1| 0|
0| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------
3| 100| 0|
0| 0| 1|
1| 1| -1|

3| 1| 0|
0| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------
100| 4| 0|
0| 0| 1|
1| 1| -1|

1| 4| 0|
0| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------
5| 100| 0|
0| 0| 1|
1| 1| -1|
```



```
5| 0| 0|
0| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------------
100| 5| 0|
0| 0| 1|
1| 1| -1|

1| 5| 0|
0| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------------
6| 100| 0|
0| 0| 1|
1| 1| -1|

6| 0| 0|
0| 0| 1|
1| 1| -1|

--------------------------------------------------------------------------------
100| 6| 0|
0| 0| 1|
1| 1| -1|
```



```
----------------------------------------------------------------------------
100| 6| 0|
0| 0| 1|
1| 1| -1|

0| 6| 0|
0| 0| 1|
1| 1| -1|

----------------------------------------------------------------------------
6| 100| 0|
0| 0| 1|
1| 1| -1|

6| 0| 0|
0| 0| 1|
1| 1| -1|

----------------------------------------------------------------------------
100| 6| 0|
0| 0| 1|
1| 1| -1|

Player wins, score =  6
```

D.References

Wikpedia: https://en.wikipedia.org/wiki/Gamification

Coursera:https://www.coursera.org/learn/gamification

Forbes:https://www.forbes.com/sites/ninaangelovska/2019/01/20/gamification-trends-for-2019-making-room-for-game-elements-in-politics/ (needs VPN)

E. Appendix

```
import random
class Game:
    def __init__(player):
        player.initialize_game()

    def initialize_game(player):
        player.current_state = [[1,-1,0],
                             [1,0,1],
                             [0,1,-1]]
        player.score = player.current_state[2][0]
        player.s = player.score
        player.start = [2,0]
        variable = player.start
        player.turn = 'player'
        player.state = [[1,-1,0],
                        [1,0,1],
                        [0,1,-1]]

    def draw_board(player):
        for i in range(0, 3):
            for j in range(0, 3):
                print('{}|'.format(player.current_state[i][j]), end=" ")
            print()
        print()

    def valid(player,r,c,x,y):
        res = player.states(x,y)
        if r < 0 or r > 2 or c < 0 or c > 2:
            return False
        item = [r,c]
        if item in res:
            return True
        else:
            return False

    def end(player, moves,goal):
        if moves == 0:
            if player.score >= goal:
                return 'player'
            else:
                return 'fail'
        return None

    available = []



def states(player,r,c):
        del player.available[:]
        if r > 0 and r <= 2 and c <= 2:
            player.available.append([r-1,c])
        if r < 2 and r >= 0 and c <= 2:
            player.available.append([r+1,c])

        if c > 0 and c <= 2 and r <= 2:
            player.available.append([r,c-1])

        if c < 2 and c >= 0 and r <= 2:
            player.available.append([r,c+1])
        return player.available
    green = []

    def max_alpha_beta(player,start,green,moves,goal,alpha,beta):
        maxvalue = -1
        result = player.end(moves,goal)
        x = start[0]
        y = start[1]
        green = [x,y]
        pre = player.current_state[x][y]
        states = player.states(x,y)
        if result == 'player':
            return(1,0,0)
        if result == 'fail':
            return(0,0,0)
        for i in range(0,3):
            for j in range(0,3):
                ss = [i,j]
            if ss in states:
                player.current_state[i][j] += player.current_state[x][y]
                player.current_state[x][y] = 100
                moves -= 1
                goal = player.current_state[i][j]
                start = [i,j]
                (minv,value) = player.min_alpha_beta(green,start,moves,goal,alpha,beta)

                if minv > maxvalue:
                    maxvalue = minv
                    x = i
                    y = j
                    start = [x,y]
                    green = start
                #prunning
                goal = pre
                moves += 1
                start = green
                for s in range(0,3):
                    for r in range(0,3):
                        player.current_state[s][r] = player.state[s][r]
                if maxvalue >= beta:
                    return(maxvalue,x,y)
                alpha = max(maxvalue,alpha)
                if alpha >= beta:
                    break
    return (maxvalue,x,y)
```

```
def min_alpha_beta(player,start,green,moves,goal,alpha,beta):
    minvalue = 2
    randomNum = [-1,0,1]
    value = None
    result = player.end(moves,goal)
    if result == 'player':
        return(1,0)
    if result == 'fail':
        return(0,0)
    x = start[0]
    y = start[1]
    randomNo = random.randint(-1,1)
    player.current_state[x][y] = randomNo
    (maxv,x,y) = player.max_alpha_beta(green,green,moves,goal,alpha,beta)

    if maxv < minvalue:
        minvalue = maxv
        value = randomNo

    for s in range(0,3):
        for r in range(0,3):
            player.current_state[s][r] = player.state[s][r]

    minvalue = min(minvalue,maxv)
    beta = min(minvalue,beta)
    if minvalue < beta:
        beta = minvalue
    if alpha >= beta:
        return(alpha,value)
        #break
    return(minvalue,value)
def play(player):
    goal = int(input('Enter minimum number to reach: '))
    moves = int(input('Enter maximum number of moves: '))
    while (True):
        player.draw_board()
        player.result = player.end(moves,goal)
        if player.result != None:
            if player.result == 'player':
                print('Player wins, score = ', player.score)
            else:
                print('fail, score = ', player.score)
            player.initialize_game()
            return

        if player.turn == 'player':
            print('---------------------------------------------------------------------------')
            while(player.turn == 'player'):
                (maxi,x1,y1) = player.max_alpha_beta(player.start,player.start,moves,player.score,-1,2)
                if player.valid(x1,y1,player.start[0],player.start[1]):
                    variable = player.start
                    player.current_state[x1][y1] += player.score
                    player.current_state[player.start[0]][player.start[1]] = 100
```

```
                        player.score = player.current_state[x1][y1]
                        player.state[x1][y1] = player.current_state[x1][y1]
                        player.state[player.start[0]][player.start[1]] = 100
                        player.start = [x1,y1]
                        moves -= 1
                        if player.score == goal:
                            player.s = player.score
                        if player.score < player.s:
                            player.score = player.s
                        elif player.score >= player.s:
                            player.s = player.score
                            player.score = player.s
                        player.turn = 'AI'

            else:
                (mini,value) = player.min_alpha_beta(variable,player.start,moves,player.score,-1,2)
                x = variable[0]
                y = variable[1]
                player.current_state[x][y] = value
                player.state[x][y] = value
                player.turn = 'player'

def main():
    g = Game()
    g.play()
   if__name__ = "__main__":
              main()
```