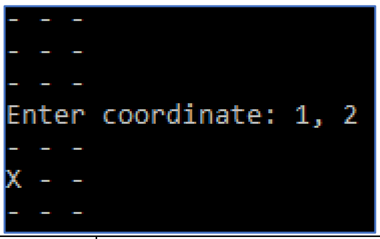
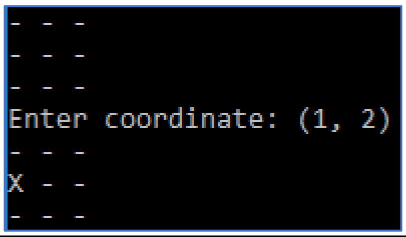
Test

During my planning, I outlined 5 test cases which should be used to verify that the final program is working appropriately. Each test case will now be tested, and the actual outcome will be provided.

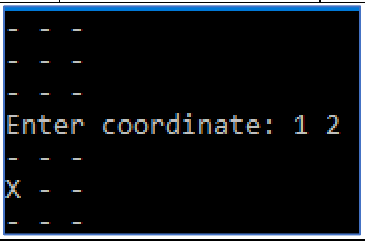
Test	Description of Test	Test Data	Expected Outcome	Actual Outcome	Comments and Fixes
1	Check whether (x, y) coordinates are accepted, and accurately interpreted.	1, 2 (1, 2) 1 2	All three forms should be accepted, and a cross should accurately be placed in the middle-left square.	All three forms are accepted, and a cross is accurately be placed in the middle-left square.	Test passed.



```
- - -
- - -
- - -
Enter coordinate: 1, 2
- - -
X - -
- - -
```

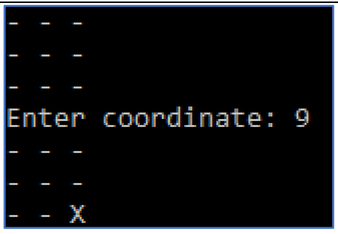


```
- - -
- - -
- - -
Enter coordinate: (1, 2)
- - -
X - -
- - -
```

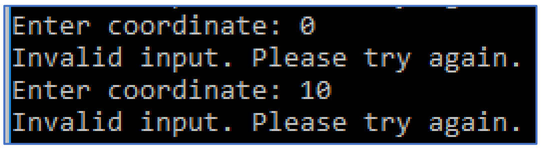


```
- - -
- - -
- - -
Enter coordinate: 1 2
- - -
X - -
- - -
```

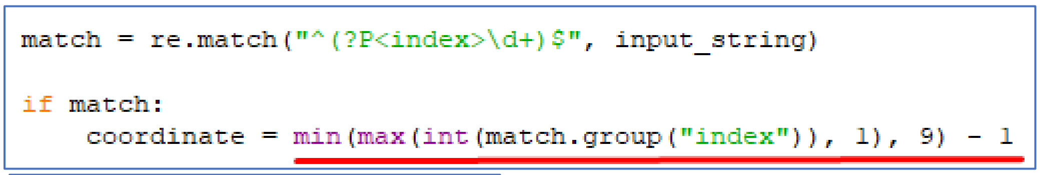
Test	Description of Test	Test Data	Expected Outcome	Actual Outcome	Comments and Fixes
2	Check whether linear coordinates are accepted, and accurately interpreted.	9	9 should be interpreted as square (3, 3) – bottom-right corner.	9 should be interpreted as square (3, 3) – bottom-right corner.	Test passed. Although coordinates in the range 1 – 10 were valid, I thought it would be a good idea to accept coordinates which are out of range, and interpret them as the edge coordinates. I implemented this by accepting user input of any digit length, and taking the min and max values in the range 1 – 9, which would truncate user input.



```
- - -
- - -
- - -
Enter coordinate: 9
- - -
- - -
- - X
```

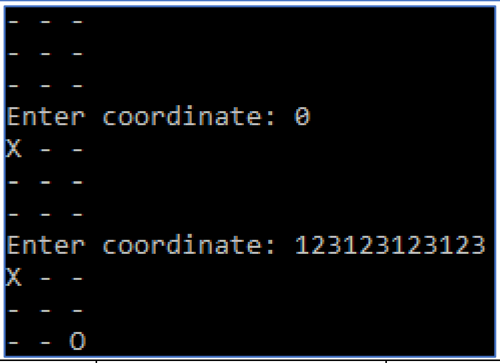


```
Enter coordinate: 0
Invalid input. Please try again.
Enter coordinate: 10
Invalid input. Please try again.
```



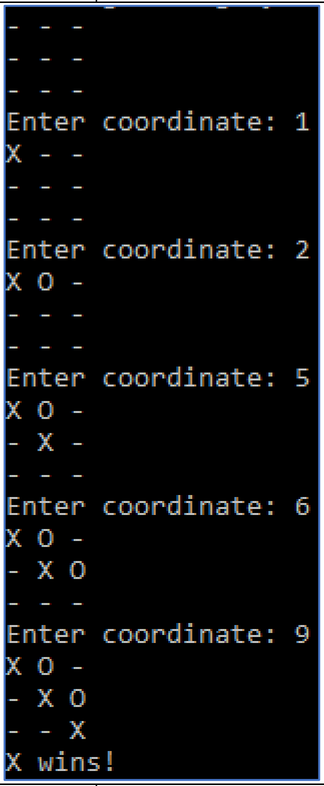
```
match = re.match("^(?P<index>\d+)$", input_string)

if match:
    coordinate = min(max(int(match.group("index")), 1), 9) - 1
```



```
- - -
- - -
- - -
Enter coordinate: 0
X - -
- - -
- - -
Enter coordinate: 123123123123
X - -
- - -
- - O
```

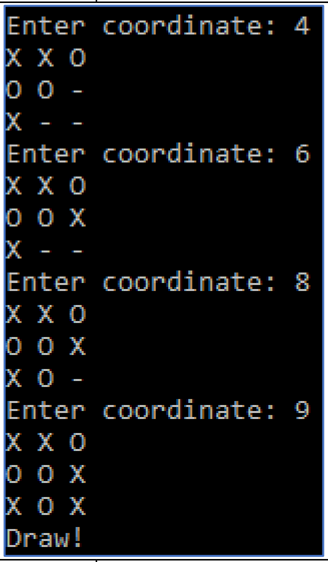
3	Check whether the game is won accurately by a player who places 3 symbols diagonally	1 2 5 6 9	First player should win.	First player ('X') wins.	Test passed.



```
- - -
- - -
- - -
Enter coordinate: 1
X - -
- - -
- - -
Enter coordinate: 2
X O -
- - -
- - -
Enter coordinate: 5
X O -
- X -
- - -
Enter coordinate: 6
X O -
- X O
- - -
Enter coordinate: 9
X O -
- X O
- - X
X wins!
```

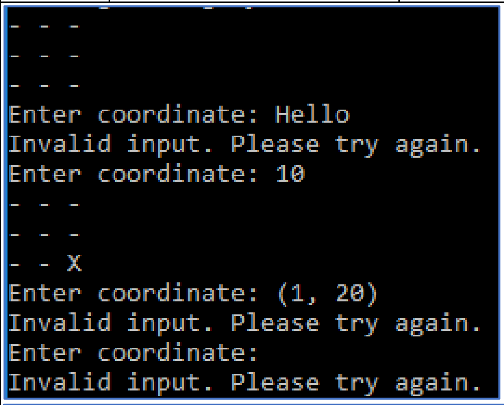
4	Check whether the game is drawn accurately when no lines of symbols match up.	1 5 2 3 7 4 6	Game should be drawn.	Game is drawn.	Test passed.

		8 9			

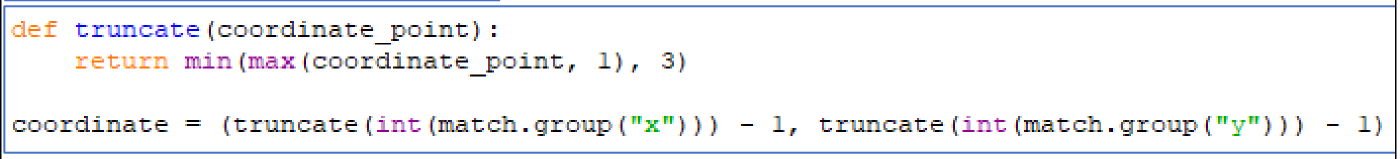


```
Enter coordinate: 4
X X O
O O -
X - -
Enter coordinate: 6
X X O
O O X
X - -
Enter coordinate: 8
X X O
O O X
X O -
Enter coordinate: 9
X X O
O O X
X O X
Draw!
```

5	Check whether invalid data is discarded.	Hello 10 (1, 20) [empty line]	User should be requested input again, and the current round will be restarted.	User is requested input again, and the current round is restarted.	Test passed. Due to the changes made in test 2, the program now accepts coordinate 10 as valid. To remedy this, I also implemented coordinate truncation for 2D coordinates, and re-tested the '(1, 20)' test – this truncated to (1, 3).

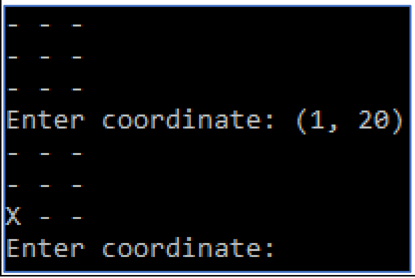


```
- - -
- - -
- - -
Enter coordinate: Hello
Invalid input. Please try again.
Enter coordinate: 10
- - -
- - -
- - X
Enter coordinate: (1, 20)
Invalid input. Please try again.
Enter coordinate:
Invalid input. Please try again.
```



```python
def truncate(coordinate_point):
    return min(max(coordinate_point, 1), 3)

coordinate = (truncate(int(match.group("x"))) - 1, truncate(int(match.group("y"))) - 1)
```



```
- - -
- - -
- - -
Enter coordinate: (1, 20)
- - -
- - -
X - -
Enter coordinate:
```