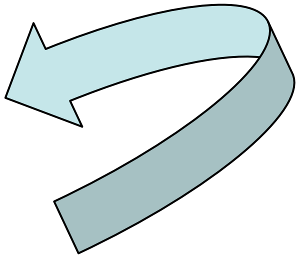
2 Dimensional Arrays



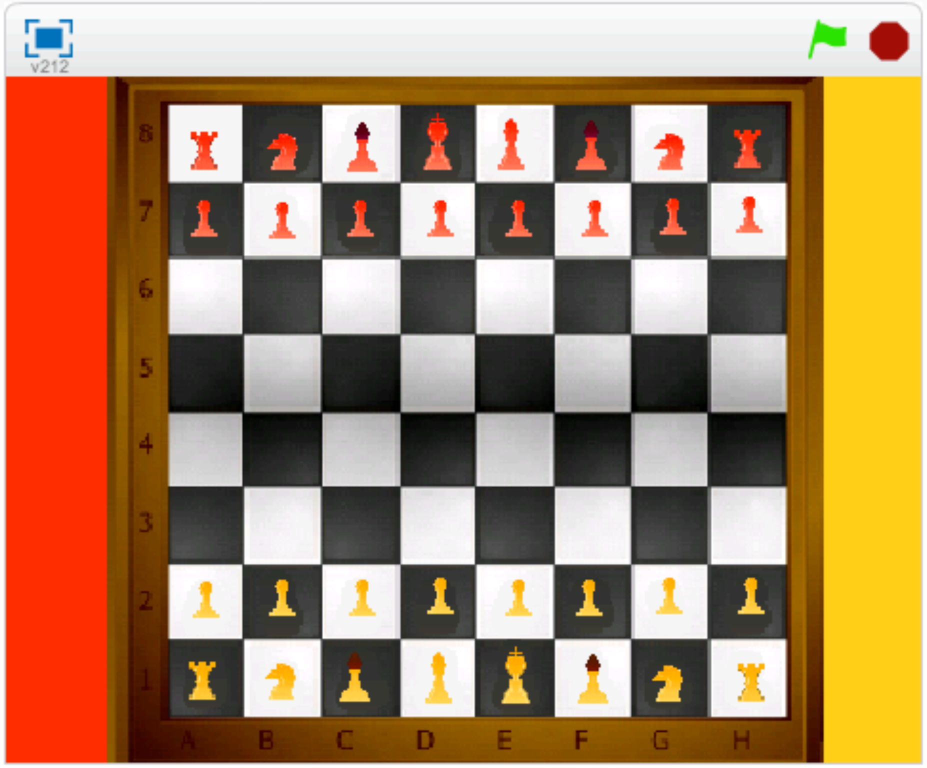
Amazing **2D Glasses**
from
http://www.freakngenius.com/

AP Computer Science, Garfield HS, Feb 2012 Earl Bergquist
Supplement presentation for AP CS based on: Building Java Programs, Chapter 7
by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/)

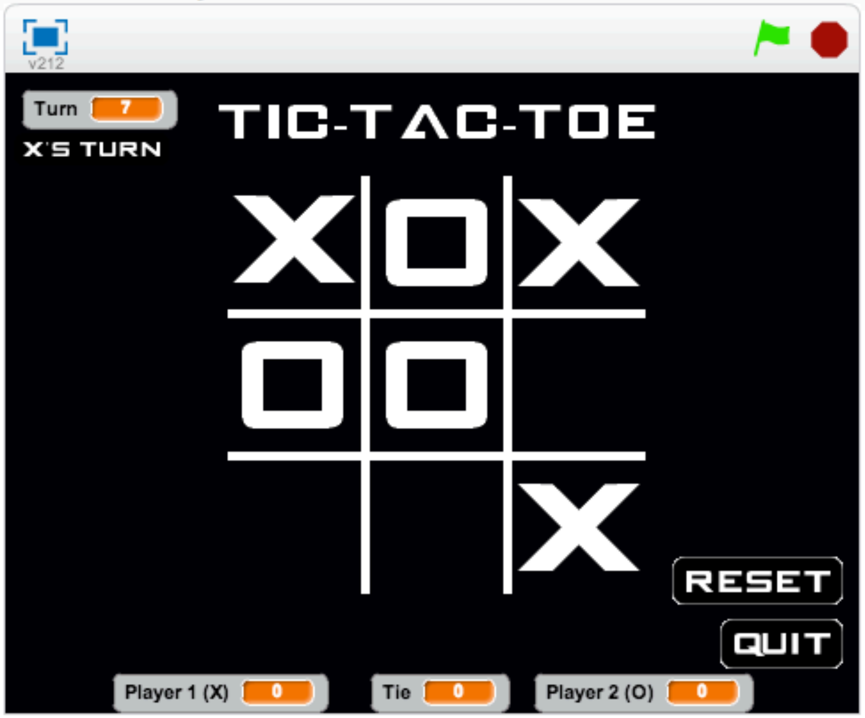
How can we use them?

- Tables of Data – as seen in MS Excel
- Represent Images
- And Grid Based Games…

chess/checkers game



Tic-Tac-Toe by Kevin



2D Arrays

Any type can be an array…

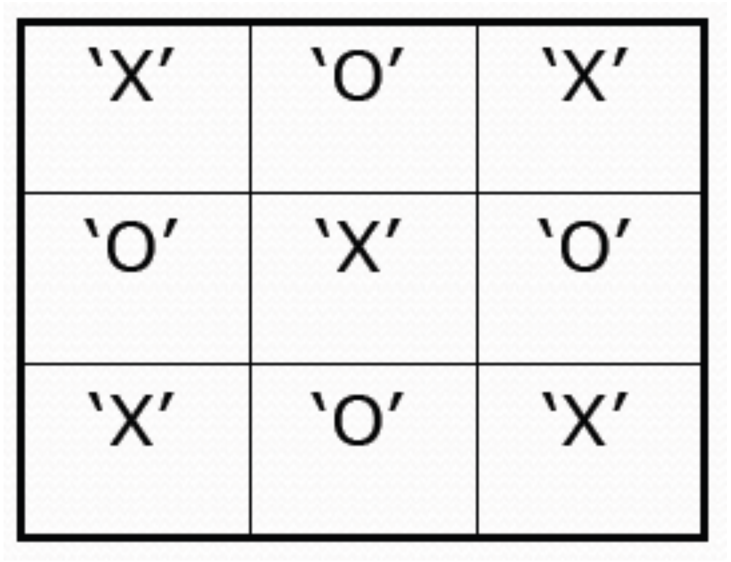
- `char[]`
- `String[]`
- `DrawingPanel[]`

And they can be a 2D Arrays

- `char[][]`
- `String[][]`
- `DrawingPanel[][]`

For example, we can make a board of char's:

`board[][]`



'X'	'O'	'X'
'O'	'X'	'O'
'X'	'O'	'X'

Declaring and initializing

Declaration similar to a single dimension Array:

```
    char[][] board = new char[3][3];
```

- Size of both dimensions must be defined
- Remember that it is filled with zero-equivalent values

Starting values can be initialized with nested { }'s:

```
char[][] board = {{'X', 'O', 'X'},
                  {'O', 'X', 'O'},
                  {'X', 'O', 'X'}};
```

This effectively creates an Array of Array's…

Sizes are defined by the length of the elements added.

Different Dimensions

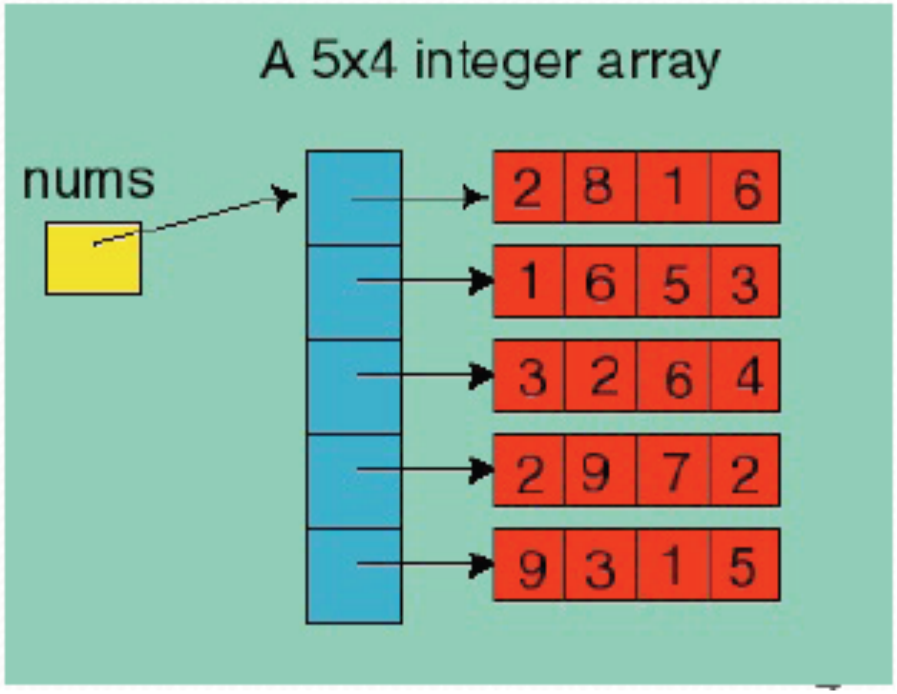
Width and height can be different:

```
   int[][] nums = new int[5][4];
```

- height is first (“an array...”)
- width is second (“...of arrays”)

i.e. `num[0][1] == 8`
(index starts with 0)
So `num[?][?]` is 7?
`num[2][2] == ?`

```
 num[3][2] == 7
 num[2][2] == 6
```



Let's try some more

```
int[][] nums = new int[5][4];
```

[5][4]	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

```
nums[0][0] = 1;
nums[0][1] = 2;
nums[1][0] = 10;
nums[3][2] = 5;
nums[2][3] = 10;
nums[4][2] = 66;
```

It can be a bit tedious.

[5][4]	0	1	2	3
0	**1**	**2**	0	0
1	**10**	0	0	0
2	0	0	0	**10**
3	0	0	**5**	0
4	0	0	**66**	0

Jagged Arrays

Not all rows in an array have to be the same length.

```
int[][] jagged = new int[3][];

jagged [0] = new int[2];
jagged [1] = new int[5];
jagged [2] = new int[4];
```

[3]	0	1	2	3	4 ...
0	0	0			
1	0	0	0	0	0
2	0	0	0	0	

Make sure to check lengths of each row before acting.

2D Array Lengths

`nums.length` - returns array's height or number of rows (no [ ])

`nums[0].length` - returns array's width or columns - at that row [ ], which is the same for regular arrays.

```
int[][] nums = new int[5][4];
```

`nums.length` is `5`
`nums[0].length` is `4`
All rows same width here.
`nums[1].length` is `4`
`nums[2].length` is `4`…

[5][4]	0	1	2	3
0	**1**	**2**	0	0
1	**10**	0	0	0
2	0	0	0	**10**
3	0	0	**5**	0
4	0	0	**66**	0

Row Width will vary for jagged arrays.

Traversing Arrays

- Acting on 2D arrays usually involves nested for loops using the length dimensions – keep track of row & columns carefully and name variables wisely.

```
// fills all elements of an int array with
// sequential values starting at 1
public static void fillArray (int nums[][]){
   int count = 1;
   for (int row = 0; row < nums.length ; row++){
      for (int col = 0; col < nums[0].length ; col++){
         nums[row][col] = count;
         count++;
      }
   }
}
```

- Will this work for a jagged array? If not how can we fix it??

  Try it with sample code ArrayOver.java:

  (http://www.garfieldcs.com/wordpress/wordpress/wp-content/uploads/2013/02/ArrayOver.java)