JavaScript Same Game

Let's design and write a complex program!

Same Game

Goal for today: write one larger program

- Watch the game
- See the progression
- We'll cover 2D arrays, then you can work at your own pace

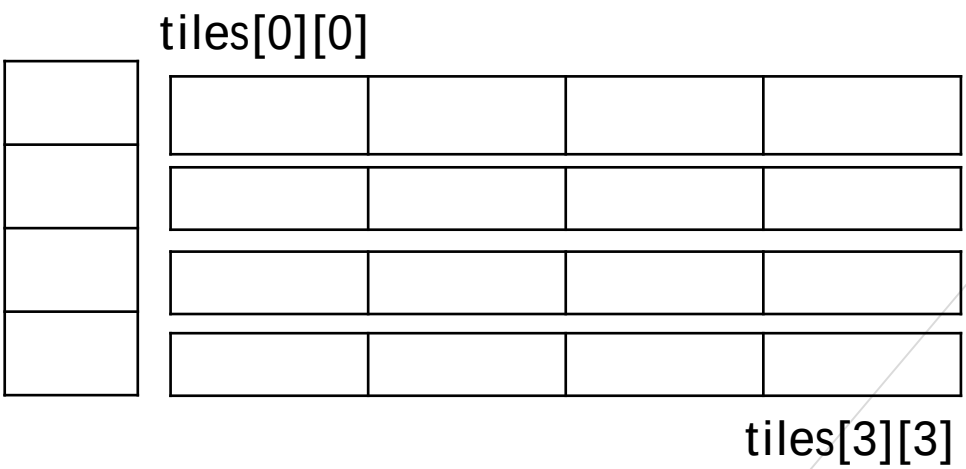
Create the Board Array

- Create 2D array with colors for each cell
- Use an array of color names (similar to Hangman words)
- A 2D array is just an array where each row is also an array

```
var tiles = new Array(numRows);
for (var row=0; row<numRows; row++) {
      tiles[row] = new Array(numColumns);
}
```

To assign colors:

- For each row
  - For each column // NESTED for loop
    - Select a color # randomly
    - Put the value in the cell:
    - `tiles[row][col] = colorNumber;`



Draw the Array on Screen

- Must convert array position to pixel location
- Must convert color number to color name
  - We could store the color name in the array, but that's a lot of memory!
- Complete the worksheet questions 1 and 2
- We'll use this worksheet again when we deal with mouse clicks

NOW PROGRAM DRAW BOARD

HINT: Use the html etc. from Hangman, adapt as needed.

Handling Mouse Clicks

- Clicking on the canvas is another type of EVENT.
- We need to add an event LISTENER to tell JS what function to call:

```
canvas.addEventListener("mouseup",
    processClick, false);
```

- "mouseup" specifies the event (there's also mousedown and others)
- processClick is the name of the function that will be called
- false just determines when event is handled (advanced topic, not necessary to understand)

When the mouse is clicked, processClick is called with an event object. To access the x/y coordinates:

```
if (event.x === undefined || event.y === undefined) {
      return; // handle error
} else {
      // parameter to processClick is event
      var xCoord = event.x;
      var yCoord = event.y;
}
```

https://www.w3schools.com/jsref/met_document_addeventlistener.asp
https://www.w3schools.com/jsref/obj_mouseevent.asp

Convert mouse x/y to array

- Mouse x/y is relative to web page
- Notice (worksheet) that the canvas is contained within the web page

```
/*
https://forums.tumult.com/t/canvas-offset-problem/11863
*/
var canvasLeft =
 canvas.getBoundingClientRect().left;
var canvasTop =
 canvas.getBoundingClientRect().top
```

- Return to the worksheet and answer question 3

Check the Neighbors

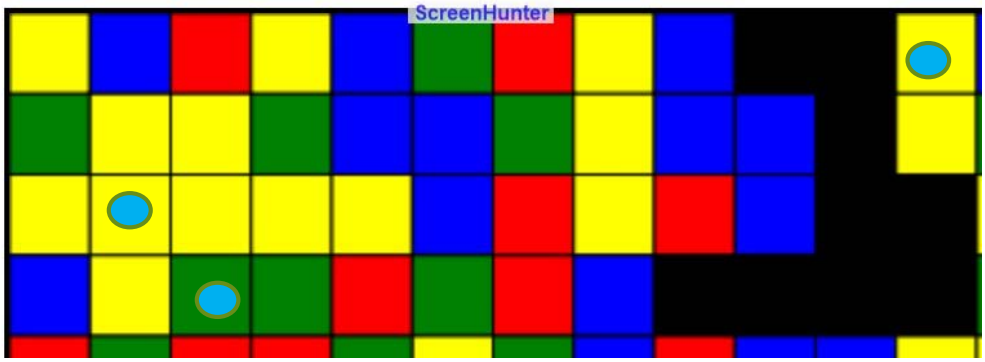
- Determine if any adjacent cells
- Adjacent is up/down/left/right – not diagonal
- Look at the worksheet. If the user has clicked on cell [1][1], what cells are neighbors?
- Assume row=1, col=1. What formula would select the cell above? Below? Right? Left?
- JS generates an error if you try to access a cell that doesn't exist. What error checking is needed?
- For now, just generate an alert with "yes" if there are adjacent neighbors, "no" if not

NOW PROGRAM CHECK NEIGHBORS

Delete Tiles

- Deleting a tile just means turning the color black (slot 0 in list of colors)
- The simplest solution to this problem is RECURSIVE – means the function calls itself
- A recursive function must have a BASE CASE – something that prevents the function from calling itself forever

Delete Tiles



deleteTiles(row, col) can be stated as

```
if this tile is black, done! (base case)
Turn this tile black
if the row above is the same color
    delete tiles above
if the row below is the same color
    delete tiles below
if the column left is the same color
    delete tiles left
if the column right is the same color
    delete tiles right
```

- Walk through (informally) how this would work for green cell in row 3, col 2 (blue dot)
- How would this work for cell in row 0, col 11 (blue dot)
- How would this work for cell in row 2, col 1 (blue dot)

Code the function

- You will simply need to convert the PSEUDOCODE (English-like description) on the former into valid JavaScript
- Be sure to include error checks (i.e., ensure you don't access row -1, etc.)
- Remember that the program has also verified that this cell has same color neighbors – so we can delete (set color to black) first, then check all of its neighbors

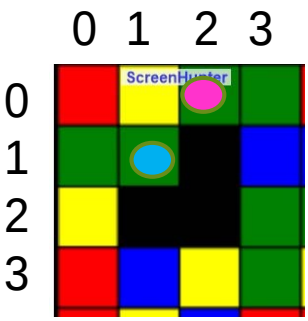
Pedagogy sidebar: Recursion is a natural way to solve certain problems. In fact, it would be much harder to code this function without it. BUT recursion in general can be a difficult topic (college sophomores often struggle). For HS, probably best to provide recursive description when it's natural, but NOT have students try to create their own (unless advanced class).

Move Tiles Down

- Note that we do an alert first, so programmer can see which cells just deleted
- Then we do the move
- "Move" means we *change the color appropriately*
  - move cell with blue dot means cell below becomes green, this cell becomes black
  - move cell with pink dot means cell two below becomes green, this cell becomes black
- We often process a grid starting at [0][0]. But in this case we need to know how many cells below are black… so we will start at the bottom of the grid [#rows - 1][0]

Algorithm:

- We process one column at a time (why col not row?)
- For each column (0..col-1)
  - set blacksBelow to 0
  - For each row, starting with [#rows - 1]
    - if this cell is black, add 1 to blacksBelow
    - else if blacksBelow > 0 // Black cells below this one
      - Move this cell down *appropriately* (see above)
      - Set this cell to black

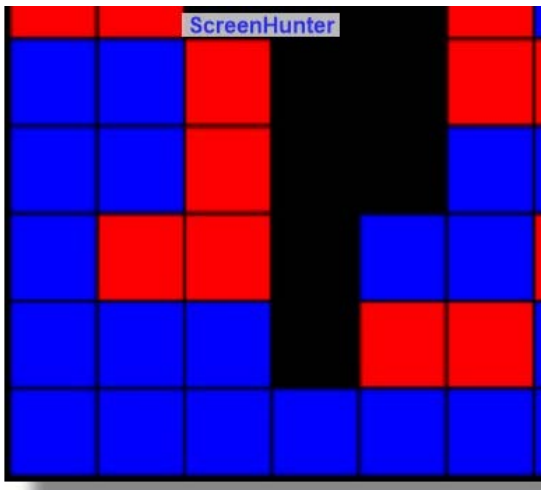


TRACE THIS on the back of your worksheet for the first 3 columns!!

Let me see your work.

Move Tiles Left

- One variable (numColors) determines the number of tile colors
- With just 2, it's quicker/easier to test moving the tiles left (happens only when an entire column is empty)
- Move Tiles Left uses a similar strategy/code as Move Tiles Down
- How do we know a column is empty? Do we have to look at every tile in that column? HINT: look at the picture below. If the user clicked a blue cell on the bottom, which column(s) would have a black cell at the bottom? If cells have been moved down, would you ever have a black cell at the bottom with colored cells above?



Finish the Game

How to score

- Set score to 0 // easy to forget!
- When deleting tiles, keep track of the # deleted
- Update score. We want to reward larger blocks of tiles. So we can use the formula:

```
score += deletedTiles * (deletedTiles – 1)
```

Game over

- If there are no adjacent cells, loss. Will need to go through every cell. If not black, then check cells above and to left. *Why don't we need to check all four directions?*
- If there are no cells remaining, win. Easy solution: just keep a running total of cells remaining.
- To report to user on web page: include a paragraph with a span that has an id, set the innerHTML of that span (your approach may vary)

End of Day 3

Let’s talk about post-work

Post Work

- Will vary by individual
- If you did not complete Hangman, finish the game
  - Maybe find another way to allow user to enter letter. Drawing boxes for the letters and responding to mouse clicks is one option.
  - You might want to display incorrect letters guessed.
- If you're looking for a challenge, pick a game to implement (maybe Minesweeper, has similarities to Same Game)
- Maybe do a jigsaw puzzle (take an image, create pieces. Display shuffled pieces. Click on image, click on grid space where you think it belongs)
- Other ideas? Just let me know. If you're applying what you've learned in an engaging fashion, it should be fine.

Appendix

Answer key for Move Tiles Down trace

Trace Example

- Doesn't need to be too formal
- Very useful technique, want to encourage students to do. Format may vary, needs to make sense to creator.
- Putting console.log in code helps us trace what computer is really doing (when we trace, we sometimes trace what we WANT it to do, rather than what we're TELLING it to do!