Pirates

Repository: https://bitbucket.org/te-cbus-cohort-11/js-wk1-review-ship-game

Starting

I start by walking through the existing parts

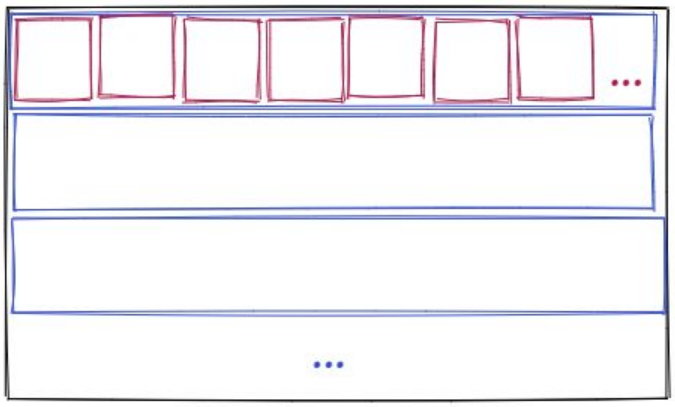
1. HTML - showing that it is minimal HTML
    a. The game will be built using DOM Manipulation in
2. CSS
    a. All the needed CSS exists
    b. I point out the CSS like .boat, .pirate, etc. that have background images that will act as the game pieces
3. JS
    a. There is some preexisting JS to speed things up that are not DOM/Event related
    b. createGrid()
        i. calls buildRow() which calls buildSquare(), these together will build the game board
        ii. The prebuilt parts are just the loops to get the parts right and the random pirate placement.
            1. You can add or subtract pirates by changing the 85 at the end of the random number. at 85, each square has a 15% chance to contain a pirate
    c. resetGame()

I then walk through the steps we will need

1. creating the grid
2. adding the pirates
3. adding the ship
4. adding the treasure
5. moving the ship
6. winning the game (ship reaches treasure)

Creating the GRID

The createGrid Method will divide the frame into 10 rows (buildRow) and each row into 10 squares (buildSquares)



While creating the Grid you can change the .row class in the css to have a border of 1px so the students can see how the rows and squares are being built. I usually turn this set to 1 while creating the grid, and then back to 0 while finishing the game.

Explain the goal with the grid, pirates, starting position of the ship, and the treasure

As I walk through the grid creation I focus on the DOM manipulation. The DOMContentLoaded event, getting the reference to the frame, creating the div elements, appending them to the container, etc.

The bolded lines are code you will need to add

1. Need the DOMContentLoaded event for the createGrid() method
```
document.addEventListener('DOMContentLoaded', () => {
```
2. Complete the CreateGrid Method
```
function createGrid() {
    const frame = document.getElementById('frame');

    for (let i = 0; i < 10 ; i++) {
        buildRow(frame);
    }
    resetGame();
}
```

3. Complete the buildRow() method
```
function buildRow(frame) {
    const row = document.createElement('div');
    row.classList.add('row');
    frame.appendChild(row);
    //removed: frame.insertAdjacentElement('beforeend', row);
    // I added appendChild above
    for (let i = 0; i < 10 ; i++) {
        buildSquare(row, i);
    }
}
```

4. Complete the buildSquare() method
   Talk about how we need to add the random pirates

```
function buildSquare(row, count) {
   const container = document.createElement('div');
   container.classList.add('square');

   if (count > 1 && count < 89) {
       if ((Math.floor(Math.random() * 100) + 1) > 85) {
           container.classList.add('pirate');
       }
   }
   row.appendChild(container);
   //removed: row.insertAdjacentElement('beforeend', container);
   // I removed above line
}
```

At this point, I change the .row to have a 0 border again so just the square board parts borders are showing

5. Build resetGame()
   Talk about how the Ship and Treasure will work with the css classes
   a. Build getShipLocation()

      This method works by getting the current ship location, which will allow us to use DOM Transversal to find the new location when moving the ship. I review what that means.

```
function getShipLocation() {
   return document.getElementById('frame').querySelector('.boat');
}
```

   b. Build the reset method

      While building this I review DOM transversal, QuerySelector vs getElementById, innerText vs innerHTML

```
function resetGame() {

   // Inform the user they can start
   const announce = document.querySelector('.announce');
   announce.innerText = "Play!";

   // Set the starting location of the boat and treasure
   const frame = document.getElementById('frame');
   frame.firstElementChild.firstElementChild.classList.add('boat');
```

```
            frame.lastElementChild.lastElementChild.classList.add('treasure');
        }
```

After resetGame is working, I refresh the board a few times until the Treasure is refreshed by a Pirate flag. This is caused by a pirate being allowed to be in the last square. I have the students figure out how to fix it and then apply the fix.

Possible fixes:

Most common student suggestion:

1) have buildSquare() check if it is the last row/count and not place a pirate (but what happens if we change the row size or decide on a different placement for the treasure?)
2) Check if the square if it contains the treasure class before adding the pirate - but treasure is applied after the pirates (*I use this to talk about classList and its methods (contains, add, remove) and the order JS code is running in*). Instead could check if the square has a pirate and remove it when adding the treasure, which will work for any board size and/or treasure placement.

```
function resetGame() {
    // Inform the user they can start
    const announce = document.querySelector('.announce');
    announce.innerText = "Play!";

    // Set the starting location of the boat and treasure
    const frame = document.getElementById('frame');
    frame.firstElementChild.firstElementChild.classList.add('boat');

    if (frame.lastElementChild.lastElementChild.classList.contains('pirate')) {
         frame.lastElementChild.lastElementChild.classList.remove('pirate')
    }
    frame.lastElementChild.lastElementChild.classList.add('treasure');
}
```

Moving the Ship

Explain how we will move the ship by using the ship's current location and its relationships in the DOM. We will create methods for each movement. I have the students decide what keys they want to control the pirates.

I have them decide on what event to use (keyup, keydown, keypress) and talk about each. I then walk through the key press event object code (https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key/Key_Values) and have them select which keys they want to use.

I also have the students suggest where we should attach the event. If we attach it to the frame, then the user has to click on the frame for them to work, but if we attach it to body then it will work regardless of what has focus when they press the keys.

During this I review the keyboard events, key codes, event listeners and handlers, why we need to add the event listener in the DOMContentLoaded event, and the event object.

The below example uses keyup and the arrow keys (which is usually the popular choice), but I modify it depending on the students event and key selection.

I usually built the event listener with method calls so I can focus on the event listener. I then go through each method we create during this and populate them with the code to move the ship.

1. Add An EventListener for the keypress, let's use the Arrow Keys

```
document.addEventListener('DOMContentLoaded', () => {
    createGrid();

    document.querySelector('body').addEventListener('keyup', (event) => {
        if (event.key === 'ArrowRight') {
            moveShipRight();
        }
        if (event.key === 'ArrowLeft') {
            moveShipLeft();
        }
        if (event.key === 'ArrowDown') {
            moveShipDown();
        }
        if (event.key === 'ArrowUp') {
            moveShipUp();
        }
    });

});
```

2. **Populate the methods to move the ship.**

   I suggest you create Left and Right first, and then up and down. This is because Left and Right are straight forward, but up and down are not.

   I have the students suggest how we can move the ship. They often suggest keeping track of the x,y coordinates and then adjusting the x/y position for where it needs to go. If they suggest this, I talk about why it is brittle and will result in a lot of extra code, and how we can take advantage of DOM Transversal to do the same thing in a way that doesn't care about the board size or have a lot of parts to keep track of.

Moving the Ship Left/Right just takes getting the next or previous Element Sibling.
To move up or down, you get the index of the ship in the children of its parent, then get its parents next/previous sibling, then apply the class to the same index on the new row.

For up and down: You can leave off setting the index to show how it spreads the ship across the row instead of placing it in a square, and have the students give you ideas on how to solve it to make sure the ship is applied to the same square that it is in the up or down row.

```
function moveShipRight() {
    const ship = getShipLocation();
    const right = ship.nextElementSibling;
    ship.classList.remove('boat');
    right.classList.add('boat');
 }

 function moveShipLeft() {
    const ship = getShipLocation();
    const left = ship.previousElementSibling;
    ship.classList.remove('boat');
    left.classList.add('boat');
 }

 function moveShipUp() {
    const ship = getShipLocation();

    let up = null;
    if (ship.parentElement.previousElementSibling != null) {
        const index = Array.from(ship.parentNode.children).indexOf(ship);
        up = ship.parentElement.previousElementSibling.childNodes[index];
    }

    ship.classList.remove('boat');
    up.classList.add('boat');
 }
```

```
function moveShipDown() {
   const ship = getShipLocation();

   let down = null;
   if (ship.parentElement.nextElementSibling != null) {
       const index = Array.from(ship.parentNode.children).indexOf(ship);
       down = ship.parentElement.nextElementSibling.childNodes[index];
   }

   ship.classList.remove('boat');
   down.classList.add('boat');
}
```

Show the problem with moving off the screen and over pirates. I have the students tell me how can we fix this? Which is by making sure the new element is not null or contain the pirate class. I use this to talk about and show how the nextElement methods return null, instead of an error, when there is no matching element.

```
function moveShipRight() {
    const ship = getShipLocation();
    const right = ship.nextElementSibling;

    if (right != null && !right.classList.contains('pirate')) {

        ship.classList.remove('boat');
        right.classList.add('boat');
    }

}

function moveShipLeft() {
   const ship = getShipLocation();
   const left = ship.previousElementSibling;
```

```
    if (left != null && !left.classList.contains('pirate')) {
       ship.classList.remove('boat');
       left.classList.add('boat');
    }

}

function moveShipUp() {
   const ship = getShipLocation();

   let up = null;
   if (ship.parentElement.previousElementSibling != null) {
       const index = Array.from(ship.parentNode.children).indexOf(ship);
       up = ship.parentElement.previousElementSibling.childNodes[index];
   }

   if (up != null && !up.classList.contains('pirate')) {
      ship.classList.remove('boat');
      up.classList.add('boat');
    }
}

function moveShipDown() {
   const ship = getShipLocation();

   let down = null;
   if (ship.parentElement.nextElementSibling != null) {
       const index = Array.from(ship.parentNode.children).indexOf(ship);
       down = ship.parentElement.nextElementSibling.childNodes[index];
   }

   if (down != null && !down.classList.contains('pirate')) {
      ship.classList.remove('boat');
      down.classList.add('boat');
   }
}
```

3. Add an Event for the Reset button - *I use this to review the event object methods: preventDefault() and stopPropagation()*

```
document.addEventListener('DOMContentLoaded', () => {
```

```
    createGrid();

    document.querySelector('body').addEventListener('keyup', (event) => {
        if (event.key === 'ArrowRight') {
            moveShipRight();
        }
        if (event.key === 'ArrowLeft') {
            moveShipLeft();
        }
        if (event.key === 'ArrowDown') {
            moveShipDown();
        }
        if (event.key === 'ArrowUp') {
            moveShipUp();
        }
    });

    document.getElementById('resetButton').addEventListener('click', (event) => {
        resetGame();
        event.preventDefault();
    });
});
```

Now move the ship from the starting position and press Reset Game. The ship will duplicate. I have the students suggest how we can fix it.

4. Update the Reset to remove the duplicate so only the new starting ship remains

```
function resetGame() {
    const ship = getShipLocation();
    if (ship != null) {
        ship.classList.remove('boat');
    }

    const announce = document.querySelector('.announce');
    announce.innerText = "Play!";

    const frame = document.getElementById('frame');
    frame.firstElementChild.firstElementChild.classList.add('boat');
    frame.lastElementChild.lastElementChild.classList.add('treasure');
}
```

Winning the Game

We need a Win condition… what does it mean to win the game?

1. Create the isWin() function

```
function isWin(nextElement) {
   if (nextElement.classList.contains("treasure")) {
       return true;
   }
   return false;
}
```

2. Create the win() method to show the Win condition

```
function win() {
   const announce = document.querySelector('.announce');
   announce.classList.add('winText');
   announce.innerText = "You Win!";
   getShipLocation().classList.remove('boat');
}
```

3. Update moveShip functions to check for the win condition when the ship is moved

```
function moveShipRight() {
    const ship = getShipLocation();
    const right = ship.nextElementSibling;

    if (right != null && !right.classList.contains('pirate')) {
        if (isWin(right)){
            win();
        }
        else {
          ship.classList.remove('boat');
          right.classList.add('boat');
        }
    }

 }
```

```
function moveShipLeft() {
   const ship = getShipLocation();
   const left = ship.previousElementSibling;

   if (left != null && !left.classList.contains('pirate')) {
       if (isWin(left)){
           win();
       }
       else {
         ship.classList.remove('boat');
         left.classList.add('boat');
       }
   }

}

function moveShipUp() {
   const ship = getShipLocation();

   let up = null;
   if (ship.parentElement.previousElementSibling != null) {
       const index = Array.from(ship.parentNode.children).indexOf(ship);
       up = ship.parentElement.previousElementSibling.childNodes[index];
   }
   if (up != null && !up.classList.contains('pirate')) {
       if (isWin(up)){
           win();
       }
       else {
         ship.classList.remove('boat');
         up.classList.add('boat');
       }
   }
}
```

```
function moveShipDown() {
   const ship = getShipLocation();

   let down = null;
   if (ship.parentElement.nextElementSibling != null) {
       const index = Array.from(ship.parentNode.children).indexOf(ship);
       down = ship.parentElement.nextElementSibling.childNodes[index];
   }

   if (down != null && !down.classList.contains('pirate')) {
        if (isWin(down)){
            win();
        }
        else {
          ship.classList.remove('boat');
          down.classList.add('boat');
       }
   }
}
```

Challenge: Extend the Game

I challenge the students to extend the game on their own after class. Here are some ideas I give them to get them started:

1. Update the reset game so it resets the pirates
2. Create a lose condition - ship hits pirate
3. Add icebergs (show the image in the img directory)
4. Make the pirates move randomly
5. Give the pirates a basic AI so they chase the ship
6. Make it so the ship can shoot a cannon at obstacles
7. Update the pirates so they can shoot a cannon at the ship
8. Update the grid to be dynamically sized to create
9. Create a scoring system
10. Create a timed game
11. Create different difficulties (easy, intermediate, advanced, torment)