SINK A SHIP, BABY!

is a new representation of a classic battleship game.

```
Sink_Ships
├──Audio
├──anchors.png
├──app.js
├──game.html
├──main.html
├──index.html
├──ships.png
├──sink-a-ship-fp1.png
├──sink-a-ship-fp1.png
├──sound.png
└──wawes.png
```

Overview

This was my first project from General Assembly's Web Development Immersive Course. It was an individual project built in a week, and was both the first proper game I had built and my first real-world type practice with JavaScript and the DOM.

Brief:

- Render a grid-based game in the browser;
- Random ship positioning;
- Scoreboard updating based on a player's actions;
- Have game options (difficulty levels);
- Design logic for winning & visually displayed;
- Include separate HTML / CSS / JavaScript files;
- Use Javascript for DOM manipulation;

- Deploy game online, using Github Pages, where the rest of the world can access it;
- Use semantic markup for HTML and CSS (adhere to best practices).

Technologies Used:

- HTML5 with HTML5 audio;
- CSS3 with animation;
- JavaScript (ES6);
- Git;
- GitHub;
- Google Fonts;
- OOP.

Approach Taken

Crete objects :

```
const ships = {
  Jessi: {
    name: 'Jessi',
    size: 2,
    coords: new Array(2),
    afloat: true,
    mark: 'j',
    color: '#95B544'
  },
  Sub: {
    name: 'Sub',
    size: 2,
    coords: new Array(2),
    afloat: true,
    mark: 's',
    color: '#654321'
  },
```

`CreateBoard()` creates the playing field (grid) with rows * cols. Creates elements on the page while simultaneously gives them x, y coordinates, and properties id loop ++

```
function createBoard() {
  const frag = document.createDocumentFragment()
  const { cols } = dimensions
  let col = 0
  let row = 0

  for (let I = 0; I < circleCount; I++) {
    const circle = document.createElement('div')
    const isEndOfRow = I !== 0 && !((I + 1) % cols) // the end is 7
    const coords = `${col}.${row}` // coordintes x.y

    circle.className = 'circle'
    circle.addEventListener('mouseover', mouseoverSound)

    circlesKeyList[I] = circle.id = coords
    circles[coords] = circle

    if (isEndOfRow) {
      row++
      col = 0
    } else {
      col++
    }

    frag.appendChild(circle)
  }

  elms.board.appendChild(frag)
}
```

Functionality

Placing ships- Collision detection

Ships are placed either horizontally or vertically:

```
if (Math.round(Math.random()) === 0) {
      direction = 'vertical'
      edge = dimensions.cols
    } else {
```

```
        direction = 'horizontal'
        edge = dimensions.rows
      }
```

Testing location 100 times :

```
function placeShip(ship) {
    let coordList = []
    let invalidPlace = true
    let placementAttemps = 0

…

if (++placementAttemps === 99) {
        console.error(`Can't place ${ship.name}!`)
        invalidPlace = false
      }
```

Picking a random circle (coordinate) to start from:

```
  function genCoords(direction, edge, ship) {
    // pick a random circle (coordinate) to start from
    const startPoint = Math.floor(Math.random() * circleCount - 1) + 1
    const startingCoord = circlesKeyList[startPoint]
    const [col, row] = startingCoord.split('.')
```

Ensuring start coord doesn't already have a ship and ship won't hang over the edge:

```
  if (
      shipLocations[startingCoord] // ensure start coord doesn't already have a s
      || endPoint + 1 > edge // ensure ship won't hang over the edge
    ) {
      return
    }
```

Design

There is a sidebar that indicates ships state and messages player about his actions

`addShipToSidebar()` renders every ship we place on the playing field sidebar (right), box by box.

`playerClick()` is the other «main function» and controls what happens when a player clicks on a box, which text pops up on the screen etc. Scenario:

1. We have already clicked the box.
2. There is a ship where -> run the help functions hitShip to check if a) we dropped the ship and b) checkWon if we won.
3. No matter what, we count down the tries with triesLeft.

If we hit a ship, we remove that coordinate from the ship's location array. When length = 0 it is sunked. All ships have key / value = float / false all ships are sunked. We've won (endGame). Otherwise, false returns. Eng Game. Different messages depending on whether we won or lost.

Audio

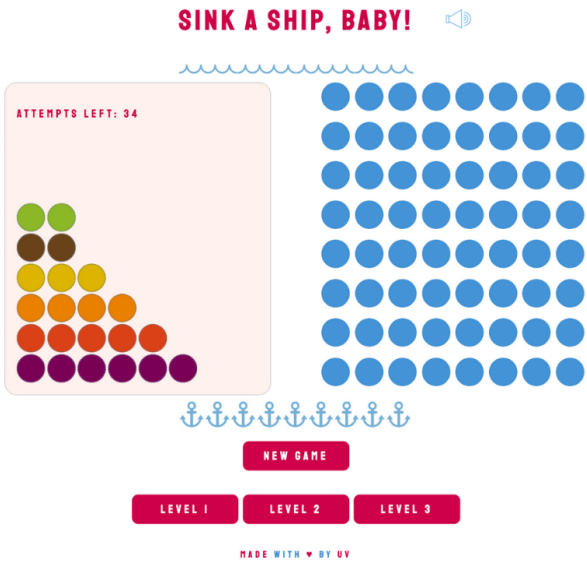
In my previous projects, I had a lot of fun working with Audio, so I added sound effects for all button clicks/presses. As a bonus, I thought it would be a good UX to enable toggling music and sounds, so I added mute-control.

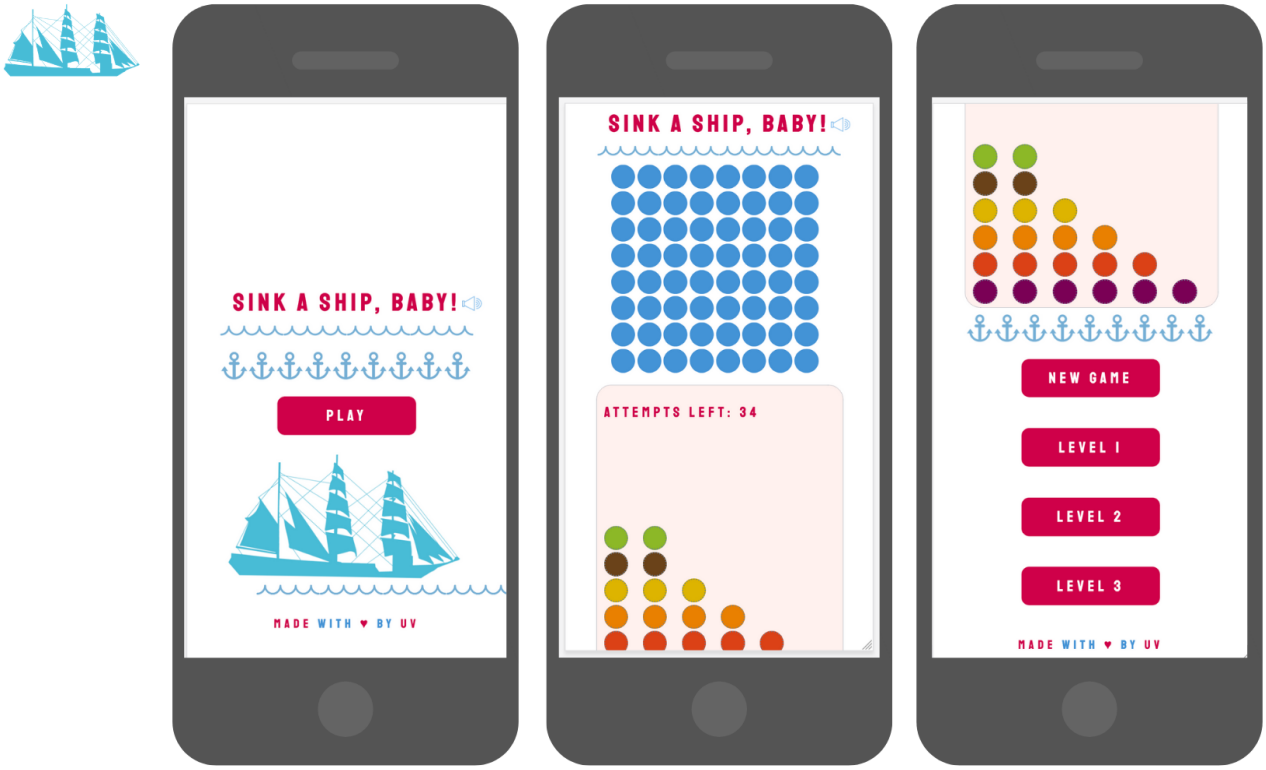
```
    elms.soundToggle.addEventListener('click', () => {
      for (const key in sounds) {
        const sound = sounds[key]
        sound.muted = !sound.muted
      }
    })
  }

// Sounds = object
```

Final Product:


SINK A SHIP, BABY!
PLAY
MADE WITH ♥ BY UV


SINK A SHIP, BABY!
PLAY
MADE WITH ♥ BY UV


SINK A SHIP, BABY!
ATTEMPTS LEFT: 34
NEW GAME
LEVEL 1
LEVEL 2
LEVEL 3
MADE WITH ♥ BY UV



Future Enhancement

There are several potential future features I'd like to implement, such as:

- More grid pattern;
- Ability to choose from a variety of color styles;
- Ability to increase the grid size;
- Authentication so users can keep track of their highest scores, compare it to other players globally.