Labo 2: Canvas animatie / interactie

Oefening 1

Maak een map aan voor deze eerste oefening en open deze in Atom.

Filestructuur aanmaken

Maak een index.html file aan en voeg een canvas tag en een script tag toe die de javascript linkt. (De script.js include plaats je vlak voor de closing body tag in je html.)

```
<canvas id="canvas" width="600" height="600"></canvas>
<script src="js/script.js"></script>
```

Rechthoeken tekenen

Definieer een globale constante met 5 kleurwaarden naar keuze:

```
const colors = [`#028a9e`, `#04bfbf`, `#efefef`, `#ff530d`, `#ff0000`];
```

Teken in de init() functie 5 balkjes met een willekeurige hoogte op de canvas.

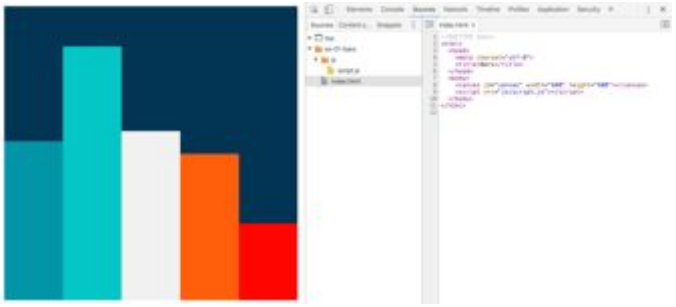
Oefening 2 : DEV TOOLS / Interactie

Chrome dev tools coding

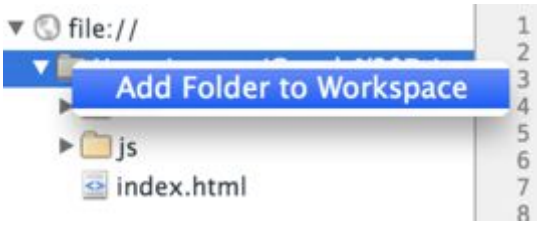
Het kan echter omslachtig zijn om telkens tussen applicaties (atom & browser) te switchen bij het aanpassen van code.

De **Google Chrome Devtools** in je browser bieden echter heel wat mogelijkheden om in Chrome zelf code edits te doen.

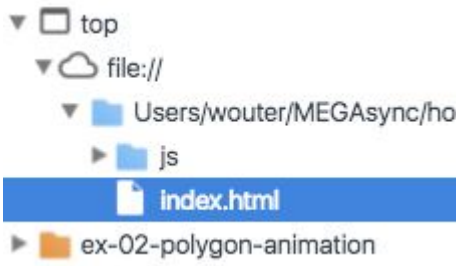
1. Dupliceer de map van de vorige oefening en open de html file in Chrome.
2. Open de Chrome Dev Tools.(View > Developer > Developer Tools)
3. Het handigst is om deze links of rechts te docken. Zo heb je voldoende ruimte om code edits te doen. Dit kun je doen met het icon naast de close button van de devtools.



4. Zorg dat de Sources tab openstaat in je dev tools.
5. Koppel de map aan het lokale filesysteem door rechts te klikken op de map in dit sources tab en te kiezen voor “Add Folder to Workspace” .



6. Chrome vraagt je toestemming, waarna je de map zal zien verschijnen onderaan in je Sources Tab:



7. (Optioneel) De volgende stap is om duidelijk te maken dat de script.js file in de file:// lokatie overeenkomt met de workspace lokatie: klik rechts op de script.js in de zopas toegevoegde map, en kies voor “Map To Network Resource”. Kies hier voor de correcte script.js in de lijst.

8. De bestanden in deze map kun je nu rechtstreeks editen in dit sources tab. Je hoeft niet te tabben naar een Code Editor. Dit kan enorm handig / snel zijn tijdens het prototypen of het uitvoeren van kleine tweaks in html / css of javascript code.

Click Interactie

Bij het klikken op de canvas, moeten de balkjes opnieuw getekend worden met nieuwe, willekeurige hoogtes. Volg hiervoor volgende stappen:

1. Definieer een nieuwe functie draw().
2. Verplaats de code die nu in de init() functie staat naar de draw functie
3. Roep de draw functie op vanuit je init functie
4. Roep de draw functie ook op bij het klikken op de canvas. (Tip: **addEventListener**)

Oefening 3: Animatie

We zullen de balkjes laten animeren naar hun doelwaarde. Hiervoor zullen we de draw functie 60x per seconde laten uitvoeren, mbv **requestAnimationFrame**.

In de draw functie, zorg je ervoor dat deze automatisch opnieuw uitgevoerd zal worden door deze te koppelen aan requestAnimationFrame:

```
requestAnimationFrame(draw);
```

Bij het uitvoeren van de code, zouden de balkjes 60x per seconde op een nieuwe waarde moeten komen. Dit komt doordat de willekeurige hoogte opnieuw gegenereerd wordt binnen de draw functie zelf.

Om te vermijden dat dit gebeurt, moeten we ervoor zorgen dat de hoogte van de balkjes niet in de draw functie berekend wordt, maar enkel bij het initialiseren van de canvas toepassing & bij het klikken op de canvas.

Waarden balkjes bijhouden

De waarden voor de balkjes zullen we bijhouden in een nieuwe, globale array. Initialiseer deze array, en vul deze op met evenveel nul-waarden als er balkjes / kleuren zijn:

```
values = [0, 0, 0, 0, 0]
```

Maak een functie generateNewValues() die deze waarden in de array vervangt door nieuwe, willekeurige waarden.

```
const generateNewValues = () => {
  values.forEach((value, index) => {
    values[index] = Math.random();
  });
};
```

Zorg er nu voor dat de applicatie hetzelfde doet als de vorige oefening (bij het klikken worden er nieuwe waarden getoond):

1. Maak gebruik van de waarden in de values array bij het tekenen van de balkjes (ipv hier Math.random() uit te voeren)
2. Roep generateNewValues() op bij het initialiseren van de applicatie.
3. Roep generateNewValues() op bij het klikken op de canvas

Animatie

We willen nu dat de balkjes niet instant verspringen naar de nieuwe waarde, maar animeren naar de nieuwe waarde.

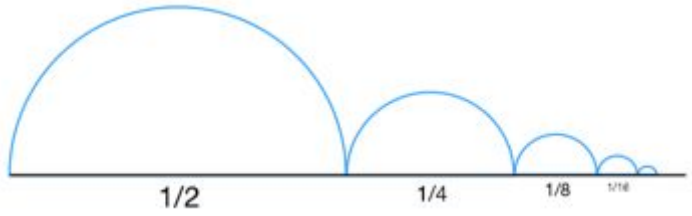
Om dit te bekomen, zullen we zowel de huidige waarde van het balkje als de doelwaarde van het balkje moeten bijhouden.

Definieer dus een tweede array met de naam displayedValues aan, die ook evenveel waarden bevat als de values array (waarin de doelwaarden zitten):

```
displayedValues = [0, 0, 0, 0, 0],
```

De hoogte van het getekende balkje zal je bepalen op basis van de waarde uit deze displayedValues array (ipv de values array). In onze draw loop zullen we de waarden in deze displayedValues array laten evolueren richting de values array.

Je kan dit op een vrij eenvoudige manier met een **easing** laten evolueren. Wanneer je van een waarde A naar een waarde B wil animeren, dan kun je dit doen door een deel van de afstand / het verschil tussen A en B op te tellen bij A:

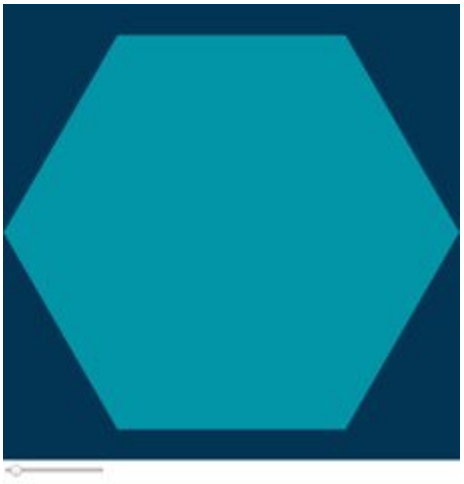


Oefening 4 - Polygon

Maak een nieuw project aan. Hierin plaats je een html file, met daarin zowel een canvas als een range input element.
Koppel een javascript bestand, waarin je de canvas, context en het range input element initialiseert.
Definieer een lege init functie die je standaard oproept. Voeg een drawPolygon() functie toe aan je code:

```
const drawPolygon = (x, y, radius, sides, color) => {
  if (sides < 3) return;
  const a = (Math.PI * 2)/sides;
  ctx.beginPath();
  ctx.save();
  ctx.translate(x,y);
  ctx.moveTo(radius,0);
  for (let i = 1; i < sides; i++) {
    ctx.lineTo(radius*Math.cos(a*i),radius*Math.sin(a*i));
  }
  ctx.closePath();
  ctx.fillStyle = color;
  ctx.fill();
  ctx.restore();
};
```

Voeg nu een draw functie toe waarin je de canvas hertekent en de polygon functie oproept. Deze draw functie voer je uit in de init functie, en bij het input event op de slider. Het aantal zijden van de polygon is gelijk aan de value van de slider.



Oefening 5 - Animated Polygon

Zorg dat de polygon langzaam roteert. Hiervoor zul je gebruik maken van een globale angle variabele en requestAnimationFrame…

Oefening 6 - Meerdere Polygonen

Breid de code uit: teken 5 polygonen die telkens 10 graden meer geroteerd staan t.o.v. elkaar, geanimeerd zijn en elk een eigen kleur hebben. Definieer de kleuren in een globale array:

```
const colors = [`#028a9e`, `#04bfbf`, `#efefef`, `#ff530d`, `#ff0000`]
```

oefening 7 - Spritesheet Animatie

Maak een nieuw project aan met daarin een html file & gelinkte javascript file. De html file bevat een canvas tag.
Maak een map assets aan en steek hier de metalslug png file in.

1. Definieer een globale constante canvas, ctx en spriteSheet en initialiseer deze. De spriteSheet constante is een Image:

```
spriteSheet = new Image();
```

2. Deze image moet de png uit de assets map inladen. Detecteer wanneer deze image is ingeladen, zoals in de theorieles getoond werd. Pas na het inladen van de image mag er een draw loop (volgende stap) starten.

3. Schrijf een drawloop (draw functie met requestAnimationFrame) die deze spritesheet toont. Bekijk de documentatie op https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage hiervoor.

Spritesheet animatie

Animaties voor games worden vaak gebundeld in 1 grote afbeelding (spritesheet). Zo moet er per frame in de animatie geen aparte image ingeladen worden.

Kijk op https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage hoe je een stuk uit deze afbeelding kan tekenen op de canvas. De frames zijn telkens 39x40 pixels groot.

Breid de code uit, zodat je elke 10 frames het volgende stuk uit de afbeelding tekent:

- Frame 0: teken vanaf x:0, y:0
- Frame 10: teken vanaf x: 39, y: 0
- Frame 20: teken vanaf x: 78, y: 0
- Frame 30: teken vanaf x: 117, y: 0
- Frame 40: teken vanaf x: 0, y: 40
- Etc…

Maak gebruik van restdeling en Math.floor om deze coördinaten te berekenen.

Keyboard control

In een game zullen we de ingedrukte toetsen op het toetsenbord uitlezen om te bepalen welke acties er uitgevoerd moeten worden.

Definieer eerst en vooral een globaal keys object. Hierin zullen we de ingedrukte keycodes bijhouden. Voorzie daarnaast ook 2 constanten voor de keycodes van de linkerpijl en rechterpijl op het toetsenbord:

```
let keys = {};
const KEY_RIGHT = 39;
const KEY_LEFT = 37;
```

In de loadedHandler koppel je de keyboard events:

```
window.addEventListener(`keydown`, keydownHandler);
window.addEventListener(`keyup`, keyupHandler);
```

In deze functies doe je een update van het keys object:

```
const keydownHandler = event => {
  keys[event.keyCode] = true;
};

const keyupHandler = event => {
  delete keys[event.keyCode];
};
```

Breidt de applicatie uit (het meeste zal je toevoegen aan de draw loop), zodat:

1. De walkcycle enkel animeert bij het indrukken van de linker of rechter pijl
2. Het karakter beweegt over het scherm bij het indrukken van de pijlen. Hiervoor kun je gebruik maken van ctx.save, ctx.translate en ctx.restore…