Roulette i 2D

Af Gregory, Oscar og Johannes

Klasse: S2o

Introduktion

Vi har fået stillet en opgave om at lave et program/spil i 2D i selvvalgte grupper, vores gruppe valgte så at lave noget som senere kunne udvikles til en gambling side, det vi har udviklet indtil videre er et roulette spil som kører over en hjemmeside, i dette spil kan du så bette på de forskellige tal og farver (rød,grøn,sort). spilleren har også muligheden for at bette på om tallet bliver lige eller ulige, samt bette på en samling af henholdsvis de første, anden, eller tredje sektion af 12 tal.

Kode opsætning

Importeret 3rd party libraries

For at gøre vores projekt nemmere at arbejde med har vi valgt at bruge forskellige 3. persons programmer (også kaldet 3rd party libraries) og frameworks til at gøre arbejdet lidt nemmer. Vi har valgt til dette projekt at lave Roulette i frameworket React sammensat med Typescript og Tailwind CSS.

React

"A JavaScript library for building user interfaces"[1]

React er helt generelt et library som gøre det nemmer at skabe et UI (User interface) med brug at komponent baseret elementer.

"Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM."

Disse komponenter er selvstændige komponenter der kontrollerer deres eget "state". Dette vil sige at alle komponenter tilkoblet til en hjemmeside bliver ikke re-renderet når du ændre på noget data. Komponentet hvor data'en bliver ændret re-render derfor kun. Et komponent bliver kun re-renderet ud fra 2 krav:[2]

1. **Parameter (aka. props) bliver ændret.**
   Når et React komponent får nyt data ind som et prop, bliver komponentet re-renderet og dataen opdateret. Der er dog nogle begrænsninger. Den ene begrænsning er hvis det nye prop forbliver det samme som det gamle prop.

2. **State af komponent bliver ændret.**
   Et komponents state repræsenterer et komponents tilhørende data. Hvis den data bliver ændret re-render komponenter.

[1] Reacts hjemmeside
[2] When does React re-render components?

En af Reacts vigtigste funktioner er “Hooks”. Disse hooks bliver brugt til fremvise og evt. ændre på data i komponentet. Det er en slags funktion til at “hook into” komponenter state og cyklus.[3]

Der er flere forskellige hooks som man kan bruge til forskellige situationer. De hooks som bliver brugt mest og som vi også kun gøre brug af er “UseState()” og “UseEffect”.

UseState hook

Eksempel på useState[4]

```
import React, { useState } from 'react'; // Import useState from React

function Example() {

 // Declare a new state variable, which we'll call "count"
 const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p> // Displays “count” of state variable
      <button onClick={() => setCount(count + 1) > // Onclick add 1 to “count”
        Click me
      </button>
    </div>
```

UseState bliver brugt generelt mest brugt til at indeholde noget data som skal kontrollere komponentens state. Eksemplet viser hvordan useState kan ændre variable “count” med tilsvarende funktion “setCount” når en knap bliver trykket på.

UseEffect Hook

Eksempel på useEffect[5]

```
function Example() {
  const [count, setCount] = useState(0);

 useEffect(() => {
   // Update the document title using the browser API
   document.title = `You clicked ${count} times`;
 });
```

[3] React Hooks
[4] React useState hook eksempel
[5] React useEffect hook eksempel

```
 return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1) >Click me</button>
    </div>
```

Eksemplet er nogenlunde det samme som i useState, dog bliver der tilføjet en "useEffect" funktion. Her bliver dokument titlen vist og opdateret opdateret med useState variabel UDEN at hele siden bliver re-renderet. Dette er hele pointen i useEffect, at udføre operationer i komponentet ud fra hvilke state komponentet er i.

Opbyggelse af useEffect

```
useEffect(() => {
  // effect
  return () => {
    // cleanup of component (gets run BEFORE effect)
  }
}, [input]) // controls when useEffect will get executed
```

Typescript

Typescript er type baseret sprog baseret på javascript. Dette gøre programmet mere sikkert for memory leaks med eksempelvis hvis int baseret variabler bliver tilsat string based variabler. Typescript skriver du typen af variablet, funktionenen eller komponentet for at holde styr på hvilke typer der kommer ind i funktionerne.[6]

Eksempel mellem Javascript & Typescript

```
// Javascript
function getRandomInt(min, max) {
  min= Math.ceil(min);
  max= Math.floor(max);
  return Math.floor(Math.random() * (max - min) + min); //The maximum is
exclusive and the minimum is inclusive
}
```

```
// Typescript
function getRandomInt(min: number, max: number) {
  min= Math.ceil(min);
  max= Math.floor(max);
  return Math.floor(Math.random() * (max - min) + min); //The maximum is
exclusive and the minimum is inclusive
```

[6] Typescript

Funktions props “min” og “max” er i Javascript ikke defineret hvilken type proppen skal være.

```
function getRandomInt(min, max)
```

Det er den dog i Typescript. Her er “min” og “max” defineret til kun at være type “number”

```
function getRandomInt(min: number, max: number)
```

TailwindCSS

TailwindCSS er et CSS framework til formål at skabe hurtig UI. Med Tailwind kan man skrive css i HTML classen isteden for at bruger de gamle ‘.css’ filer. Forskellen på Tailwind og den normale måde at gøre det på vil se sådan ud.

```
 <h2 class="m-5 p-2 font-bold text-white">Enter text here</h2>
```

```
h2 {margin: "1.25rem"; padding: "0.5rem"; font-weight: bold; color:"white";}
```

File tree

Opsætning af projektet kan vises ud fra et såkaldt “File tree”. Sådan et diagram har til formål at skabe en nem og overskuelig visualisering af projektets struktur. Opsætningen af projektet er inddelt i 2 generelle folders. “components” og “logic”. “components” er alle komponenter for projektet som bliver sat sammen til et sidste output i filen ”‘index.tsx”. Logic er alt de logistikken bag projektet. Hjerne af projektet der består af forskellige funktioner som bliver importeret i alle “components” filerne. Diverse andre filer i Roulette folderen er figurationer til typescript kompileren og TailwindCSS.

```
📦Roulette
├ 📂src
|  ├ 📂components //Components of the project
|  |  ├ 📜BetTable.tsx //Table for bet buttons
|  |  ├ 📜Betting.tsx //Management of assets and bets
|  |  ├ 📜Button.tsx // Custom button for project
|  |  ├ 📜History.tsx // History of rolled results
|  |  ├ 📜Option.tsx // Option page
|  |  ├ 📜Tiles.tsx // Array of Roulette tiles
|  |  ├ 📜Wheel.tsx // Roulette wheel
|  |  └ 📜Winner.tsx // Winner pop-up
|  ├ 📂logic // Logical functions for project
|  |  ├ 📜Data.tsx // Data between bet and result
|  |  └ 📜Renders.tsx // Rendering of components
|  ├ 📂static // img etc.
|  |  └ 📜Preview.png
|  ├ 📜index.css
|  ├ 📜index.tsx  // Root of project
|  └ 📜react-app-env.d.ts
├ 📜.gitignore
├ 📜craco.config.js
├ 📜package-lock.json
├ 📜package.json // Dependencies
├ 📜README.md
├ 📜tailwind.config.js // TailwindCSS config
└ 📜tsconfig.json  // Typescript config for compiler
```

Teori

Variabler

En variabel er pladsholder for et stykke data. Typisk set bruger man variabler når man skal ændre på en værdi under programmets køretid. for eksempel hvis man gerne vil have en bold til at bevæge sig fra højre til venstre på skærmen kunne det være en god ide at give den x-koordinat en variabel så man kan ændre på den istedet for at manuelt indsætte en ny x-værdi hvert 1/10 sekund.
Her er et eksempel hvor vi har lavet 3 forskellige variabler som vi i en helt masse forskellige funktion som gør dem mere læs bare så der ikke står et tal hele tiden og vi skal huske hvad hvert tal gør så bruger vi navne så vi kan huske hvad de gør.

```
// Praktisk eksempel på variabler
// (Sprog: typescript)

const RoolTime: number = 7500;
//default: 7500
const IntermissionTime: number = 9500;
// default: 9500
const CursorDisabledTime: number = 12000;
// default: 12000
```

Betinget udførelse (If statement)

Betinget udførelse eller if statement, er et stykke kode som checker om noget opfylder nogle specifikke kriterier. hvis de kriterier bliver opfyldt ville koden inde i dette if statement blive kørt. med en if statement kan man give den flere kriterier ved at lave en “else if” eller “else” statement efter den første stykke kode. “else if” er et til if statement med en betingelse før den bliver kørt, samtidig med at dette stykke kode kun bliver kørt hvis det første if statement ikke opfylder sin betingelse. en “else” statement er bare et stykke kode som vil blive kørt hvis koden ovenover ikke opfyldte sin betingelse. det kunne se ud på følgende måde:
Her bruger vi if statements til at finde ud af hvem der har vundet og hvem der haft tabt.

```
// Praktisk eksempel på en If statment


function IfSetBalance(
arg: boolean, multiplyer: number, feedback:
string) {
    if (arg) {
setBalance((balance += bettingAmount *
multiplyer));
      console.log(`You win on ${feedback}`);
    } else {
setBalance((balance -= bettingAmount));
      console.log(`You lose on ${feedback}`);
    }
}
```

Løkker

En løkke er et stykke kode som vil fortsætte med at køre indtil en given betingelse er opfyldt. der er to slags løkker, “while” - løkker og “for”-løkker. while løkker kører mens betingelsen i parentesen bliver opfyldt. mens for løkker modtager en variabel, som i det simpleste tilfælde vil tælle op eller ned, ift hvor mange gange man vil have sin løkke skal køre. løkker er meget brugbare siden det stopper en fra at skulle skrive 10000 linjer af den samme kode hvis man skal køre noget flere gange.

Her har du et eksempel på en løkke som vi bruger til at regne ud om du indsætter et bet med flere penge end du har.

```
// Praktisk eksempel på en løkke

let quickBets = [10, 25, 50, 100, 250, 1000];
//sets values of buttons

for (let i = 0; i < quickBets.length; i++) {
    <Button
      color="blue-700"
      onClick={() =>
     {
if (bettingAmount + quickBets[i] <= balance)
{setAmount((betAmount += quickBets[i]);
      }
else alert("Sorry you are over the limit!");
        }}
      >
        {quickBets[i]} $
    </Button><
  );
}
```

Lister/Arrays

Et array er en datastruktur, der indeholder en gruppe af elementer. Disse elementer er typisk alle af samme datatype, såsom et integer eller en string. Arrays bruges typisk set i computerprogrammer til at organisere data, så et relateret sæt værdier nemt kan sorteres eller søges.

Her har vi lavet et array over de forskellige farver og deres tal som vi så bruger til at indsætter dem på hjemmesiden.

```
// Praktisk eksempel på et array

const Tiles: Array<TileType> = [
  {
    val: 0, //value of tile
    color: "green", //color of tile
    pos: 1160,//position of tile used by the wheel
  },
  {
    val: 32,
    color: "red",
    pos: 1208,
  },
  {
    val: 15,
    color: "black",
    pos: 1256,
  },
...
```

Switch[7]

En switch minder lidt om et if statement, en switch får et input og derefter checker om dette input passer til de cases inde i switchen. disse cases er ligesom else if statements i en if statement, men en switch checker alle disse cases igennem samtidig i stedet for at gøre det en efter en som gør at programmet kører hurtigere

```
// Praktisk eksempel på en Switch
switch (btnId: string) {
/* Cases to compare bet with result. Returns setBalance() with
negativ or positiv bettingAmount
(maybe include a multiplyer if bets right on green or one of
the twelve) */
    case "table":
     IfSetBalance(bet.includes(result), 14, "table");
      break;
    case "green":
      IfSetBalance(bet.includes(result), 14, "green");
      break;
    case "red":
    case "black":
     IfSetBalance(result.color === bet[0].color, 1, "color");
      break;
```

[7] Switch

Array.prototype.map()[8]

Map()-metoden opretter et nyt array, der er udfyldt med resultaterne af at kalde en forudsat funktion på hvert element i det kaldende array.

```
// Praktisk eksempel på .map() funktion

Tiles.slice(1).map((value,index) => ( // gets all Tiles except 0
  <Button
    onClick={() => {
       setBet([value]); //sets users bet
       setBtnId("table"); //sets clicked btn
    }}
    key={index}
    color={value.color}>
  {value.val} //sets Btn child to current val of .map()'s value
 </Button>)
```

Array.prototype.include()[9]

kan man bruge til at finde ud af om ens array indeholder en bestemt værdi som kan være et tal eller en string. include() returnerer en boolean som holder værdien true eller false.

```
...
case "green":
  IfSetBalance(bet.includes(result), 14, "green");
 break;
...
```

Analyse

Projektet er relativt komplex i form af forskellige React states som bliver fordelt igennem de forskellige komponenter og funktioner. Disse states tag ofte også typen TileType indefra "Tiles.tsx". "Index.tsx" er hoved grenen af hele programmet. Disse states bliver indsat i de forskellige komponenter gennem props.

```
const [balance, setBalance] =
useState<number>(10000);
//start balance
const [result, setResult] =
useState<TileType|undefined>();
//result of rolled Tile
const [bet, setBet] = useState<undefined |
Array<TileType>>(); //users bet
const [btnId, setBtnId] = useState<string |
undefined>();
//button id for the clicked button for users bet
```

[8] .map()
[9] .Include()

En af disse komponenter som indtag en af disse states er “History.tsx”

“History.tsx” indtag result som prop fra “index.tsx” typen er TileType eller undefined.
`History: FC<{ result: TileType | undefined }> = ({ result })`
Grunden til at result kan være undefined er da der er op til et andet komponent (Wheel.tsx) at declare result inden de andre komponenter kan bruge result.

“History.tsx” har også sin egen state som er til for at indeholde dataen fra forrige resultater.
`const [history, setHistory] = useState<Array<TileType>>([]);`
Her kommer endnu en React hook til brug, useEffekt. Den har som effekt at hver gang der er et nyt “result”, indsætter den “result”’s nye værdi til “history”’s state.
`setHistory((prev) => [result].concat(prev));`
Da komponentet nu bliver re-renderet da den har en ændring i den prop, printer den nye array af “history” staten:

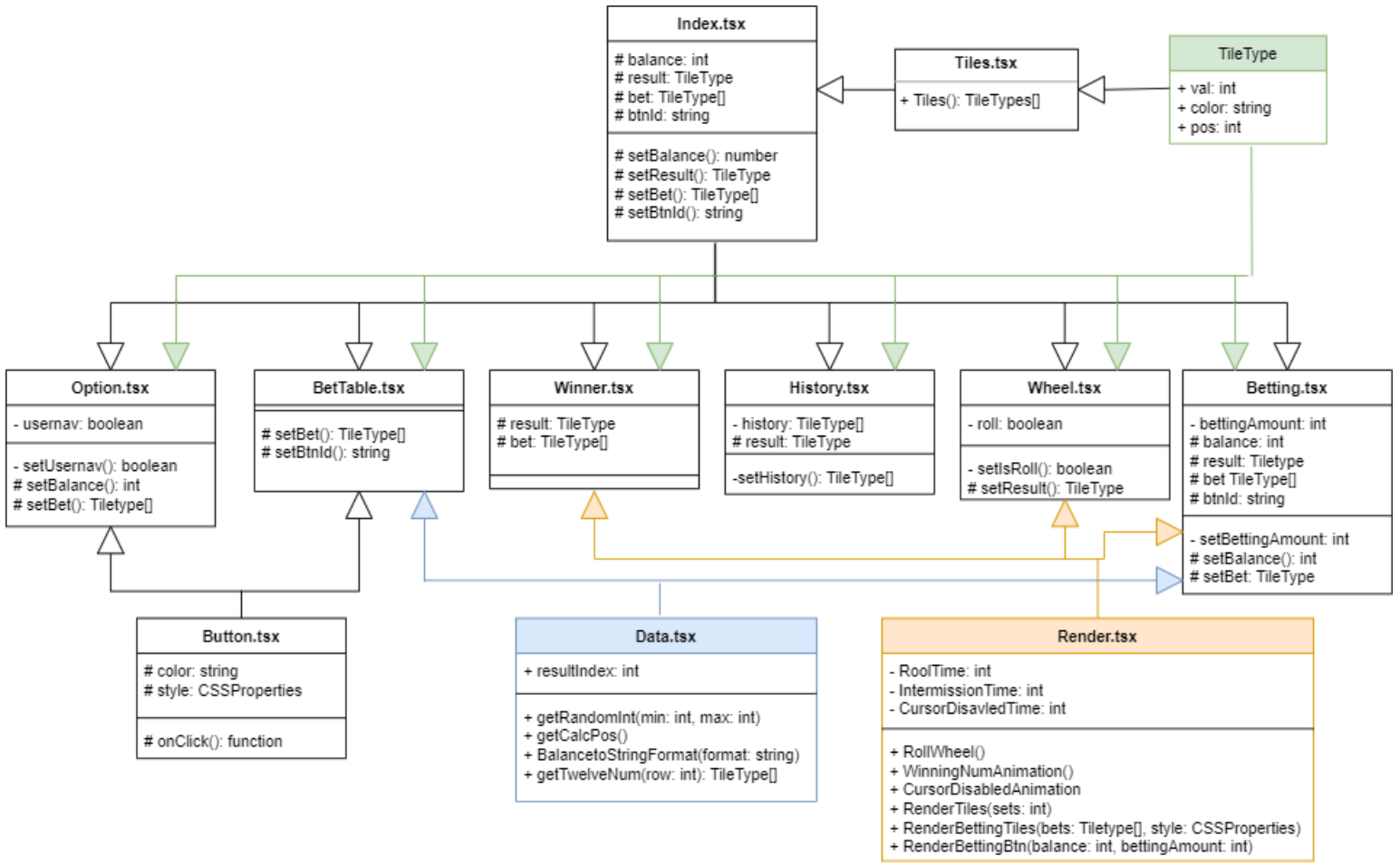
```
{history.map((value, key) => {
  //maps through history array
  return (
    <li
className={`bg-${value.color}`}
key={key}>
      <span>{value.val}</span>
    </li>
         );
       })}
```

```
//History.tsx
import { TileType } from "./Tiles";
import { FC, useState, useEffect } from "react";

const History: FC<{ result: TileType | undefined }>
= ({ result }) => {
  const [history, setHistory] = useState<Array<TileType>>([]);
  useEffect(() => {
    if (result !== undefined) {
      setHistory((prev) => [result].concat(prev));
//concats new result with prev result
    }
  }, [result]);  //Effect executes when gets new result

return (
 <aside>
    <ul>
       {history.map((value, key) => {
          //maps through history array
          return (
            <li
              className={`bg-${value.color}`}
              key={key}
            >
              <span>{value.val} </span>
            </li>
          );
       })}
      </ul>
    </aside>
  );
};
```

Projektet handler generelt om hvordan disse React states bliver fordelt til de forskellige komponenter som props. Diverse komponenter bruger også forskellige funktioner til både logistik og rendering af elementer. Disse funktioner bliver importeret fra “ Data.tsx” eller “Renders.tsx” til at gøre projektet mere overskueligt, da der er mange funktioner som bliver genbrugt i forskellige komponenter.

For en nemmer forståelig struktur af projektet, se bilaget af UML (class diagram). På diagrammet kan ses dataflowet fra komponent til komponent.

Class Diagram (UML)



Class diagram (UML) over Roulette projektet og diverse komponenters dataflow til hinanden.
Bemærk! Det er IKKE forskellige classes men React komponenter der bliver henvist til.

Diskussion

Vores produkt blev godt, men det kunne sagtens forbedres, selve spillet fungerer godt for det meste, men der er en lille fejl, hvis man får det samme resultat to gange i streg ville resultatet ikke blive talt på grund af at useEffekt hooken. Den hook styrer logistiken bag programet, og bliver ikke opdateret når det samme prop bliver indsat. Dette er jo faktisk et stort problem hvis vi havde tænkt os at udgive denne gambling side til det offentlige, er der mange som vil være utilfredse. oven i det kunne vores hjemmeside udvides til ikke kun at have roulette men et fuldt ud gambling site med blackjack, poker og slots, og muligvis andre spil. dette kunne måske gøres lettere i et andet sprog end javascript, eller på en anden platform, altså ikke en hjemmeside men måske et steam spil eller lignende.

Perspektivering/Opsummering/konklusion

Projektet om at lave et program i 2D lykkedes og vi vil selv mene det ikke blev helt dårlig, vi endte med et funktionelt spil roulette, det eneste vi mangler er nogle få bug fixes og så en deposit knap så der kan blive indskudt penge. Der er selvfølgelig en masse konkurrence siden roulette er rimelig populært, og andre hjemmesider har gjort et stort stykke arbejde for at lave deres spil til perfektion. Så vores spil kan ikke rigtig konkurrere på det niveau endnu, men det kan det basale og i fremtiden kunne det nok komme op på niveau med de store gambling sider. hvis vi virkelig ville skabe konkurrence skulle vi nok droppe hjemmeside ideen og måske rykke over til en app, eller nærmere spil siden der er langt mindre af det.

Litteraturliste

A JavaScript library... [React]. (s.d.). React. Lokaliseret den 09. Januar 2022 på https://reactjs.org/

Array.prototype.includes(). (s.d.). developer mozilla. Lokaliseret den 09. Januar 2022 på https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/includes

Array.prototype.map(). (s.d.). developer mozilla. Lokaliseret den 09 . Januar 2022 på https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

felix , gerschau. (s.d.). When does React re-render components?. felix gerschau. Lokaliseret den 09. Januar 2022 på https://felixgerschau.com/react-rerender-components/#when-does-react-re-render

React Hooks. (s.d.). javatpoint. Lokaliseret den 09. Januar 2022 på https://www.javatpoint.com/react-hooks

switch. (s.d.). developer mozilla. Lokaliseret den 09. Januar 2022 på https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch

TypeScript is JavaScript with syntax for types.. (s.d.). TypeScript. Lokaliseret den 09. Januar 2022 på https://www.typescriptlang.org/

Using the Effect Hook. (s.d.). React. Lokaliseret den 09 . Januar 2022 på https://reactjs.org/docs/hooks-effect.html

Using the State Hook. (s.d.). React. Lokaliseret den 09. Januar 2022 på https://reactjs.org/docs/hooks-state.html