Elias Soprani 20011210-8618
Linnéuniversitetet, 1ME325, Webbteknik 5 VT22, 2023-06-01.

1ME325: Webbteknik 5 Inlämnigsuppgift - *restinlämning*

Lösningsförslag:

Main klassen: Main är en klass skriven i en objekt literal struktur. Enligt Rollbar (2021) rekomenderas den objekt literala metoden då man ska skapa endast en instans av klassen. Constructor metoden används för Dice och diceApp klasserna då de är klasser som ska instanseras flera gånger. Main är den klass som startar programmet och hanterar programmets applikationfönster.

Main klassen har inga egenskaper då Main klassens huvudsyfte är att

Klassen innehåller också metoderna:

- **Init**(): Metoden anropas när programmet laddas via en händelselyssnare som sitter fäst på det globala window objektet. Metoden ansvarar för att applicera en “click” händelselyssnare på icon-dice elementet som anropar Main.createobjs metoden.
- **createobjs**(): Metoden ansvarar för att skapa en ny DiceApp instans och anropa dess createDOM metod.

Main klassen ansvarar för att instansiera DiceApp’s applikationfönster.

DiceApp klassen: DiceApp klassen ansvarar för att generera ett enskilt applikationsfönster och applicera all stilsättning, händelsehanterare och metoder relevanta för ett enskilt applikationsfönster. Klassen har många egenskaper för att hålla referenser till DOM elementen så att varje metod i klassen inte behöver göra var sin document.getElementByClassName metod för att manipulera önskade DOM element. Detta bidrar till reducerad kod redundans. De egenskaper klassen har som inte endast används för att referera till specifika DOM element är:

- **dragStartX:** Egenskapen håller ett värde som representerar var på ett applikationsfönster användaren har tryckt i X-led
- **DragStartY:** Egenskapen håller ett värde som representerar var på ett applikationsfönster användaren har tryckt i Y-led
- **diceOnBoardValue**: Egenskapen håller en siffra som representerar det sammanlagda värdet av alla tärningar som blivit kastade.
- **diceOnBoard**: Är en array som håller varje skapad Dice instans. Egenskapen används i klassens metoder remove, add och roll. För att i dessa metoder anropa antingen metoderna “destroy” eller “roll” i en enskild Dice instans. Detta innebär att det t.ex. Är möjligt i DiceApp.remove anropa “.Destroy()” metoden i den senaste Dice instansen för att ta bort den Dice instansen.
- **audio**: Egenskapen instansierar ett nytt Audio objekt. Audio objektet returnerar, enligt Mozilla (2022), ett HTMLAudioElement och tar emot en parameter av en URL till önskad

ljudfil som ska användas av det returnerade HTMLAudioElement objektet. Metoden .play() används på Audio objektet för att spela upp specifiserad ljudfil.

- **diceClasses**: Egenskapen håller en 7 index lång array där det första indexet är tomt och resterande består av strängar som representerar olika klassnamn för olika tärningssidor.

I konstruktorn för klassen sätts även this.mouseMove och this.mouseUp till "this.mouseMove.bind(this)" och "this.mouseUp.bind(this)."".bind()" returnerar metoden "this.mouseUp/Move" med ett modifierat 'this'-värde som refererar till DiceApp istället för att referera till eventet som skett när dessa metoder anropas (Mozilla, 2023). Den returnerade metoden från ".bind()"-metoden kopieras till mouseDown, mouseMove samt mouseUp. Detta är nödvändigt för att om metoderna "this.mouseUp/Move" inte ändrats till de modifierade metoderna returnerade av ".bind()" så kan inte eventlyssnare avregistreras av "removeEventListener" som har mouseMove och mouseUp som eventhanterare.

Metoderna klassen använder är:

- **add**(): Metoden används för att generera en ny tärning och fästa den till applikationsfönstret. För att begränsa att för många tärningar kan genereras så sker en beräkning som räknar ut vad max antal tärningar som får plats inom contentWrapper är, baserat på hur bred och hög contentWrapper är. En tärning har bredden 33 och höjden 34 med margin är inräknat. Detta tillåter att applikationfönstrets bräde kan ha olika storlekar och programmet skulle ändå inte tillåta att fler tärningar får genereras om det skulle resultera i att en tärning inte kommer få plats grafiskt på brädet. T.ex: När contentWrapper har höjden och bredden x = 335, y = 137. Med beräkningen = (Math.floor(x / 33)) * (Math.floor(y / 34)) blir antalet tärningar 40. Om contentWrapper istället hade x = 421 och Y = 161 så blir max antal tärningar 48. Oavsett storleken på brädet kommer antalet tärningar justeras och få plats inom brädet utan att gå över sidorna av fönstret. Math.floor används på både x/34 och y/33 och på summan av de två kvoterna multiplicerade med varandra. Detta är korrekt då siffrorna måste rundas ned. Om antalet blivit t.ex. 41.2 eller 41.9 så kan inte en tärning vara 0.2 eller 0.9 utan måste alltid ett heltal. De måste alltså rundas ned då max 40 tärningar kan visas utan att fördärva gränssnittet. Ett problem med denna lösningen är dock att denna beräkningen måste ske innan en tärning har skapats. Då måste man använda en förbestämd höjd och bredd på en tärning. Alternativet skulle vara att man först instansierar en tärning en enstaka gång. Beräknar bredden och höjden på tärningen och sparar värdena och sedan ta bort tärningen för att sedan använda metoden som vanligt. Denna lösningen skulle innebära att för att räkna ut max antal tärningar skulle tärningens propotioner tas i åtanke, men det skulle också medföra att metoden beter sig på ett sätt första gången den används, och sedan på ett annats sätt nästa gång den används och två extra egenskaper skulle behöva skapas. 1. Bredd på tärning, 2. Höjd på tärning. Jag valde då att implementera den första lösningen då det är större sannolikhet att endast applikationfönstret skulle kunna ändra storlek, än att både applikationstorleken och bildfilsstorleken för en tärning ändras. När beräkningen har utförts används en if-sats för att kolla så att mängden tärningar på brädet inte överskrider max antal tärningar som får finnas. Om if-satsen är sann så instansieras en ny instans av Dice klassen och anropar .add() metoden i Dice klassen. Den instansierade Dice instansen har då fått ett värde efter .add() metoden och kan hämtas genom att kolla instansens egenskap "value" genom att skriva: "dice.value". Ett <li> element skapas med klasserna "dice" och ett klassnamn som motsvarar vilket värde tärningsinstansen har så att tärningen grafiskt visar korrekt sida. Sedan sätts <li> elementet som child till ulForContent egenskapen som representerar ett <ul> element. Sedan adderas den instansierade tärningens värde till diceOnBoardValue och värdet av tärningen tilläggs i

diceOnBoard arrayen. Metoden displayScore() anropas sedan för att uppdatera poängräknaren. Sedan anropas audio.play() för att spela upp ett ljud.

- **remove**(): Remove metoden ansvarar för att ta bort en tärning och räkna om det sammanlagda värdet av tärningarna på brädet. I början av metoden körs en if-sats som kollar om diceOnBoardValue inte är 0, då om det är noll innebär det att det inte finns några tärningar på brädet och ingen tärning att ta bort. Om värdet inte är 0 så subtraheras först den senaste Dice instansens värde från diceOnBoardValue. Sedan anropas den senaste Dice instansens metod "destroy" för att ta bort Dice instansen så att Garbage collectorn hämtas upp den. Sedan används .splice() metoden på diceOnBoard egenskapen. .splice() är en metod som enligt Mozilla (2022) ändrar innehållet i en array genom att ta bort ett element från ett specifierad index i arrayn.
- **roll()**: Roll är en metod med uppgift att ta läsa av antalet tärningar på brädet, nollställa poängräknaren, anropa Dice.roll() metoden i varje enskild Dice instans och sedan uppdatera poängräknaren. En array skapas från ulForContent's children, dvs, <li> elementen som representerar tärningarna. Detta görs genom Array.from() metoden. Array.from() metoden kan enligt Mozilla (2022) returnera en array från ettt "itererbart eller array liknande objekt". Returen från Array.from() lagras i graphicDiceArray. Efter detta körs en for-loop för så många Dice instanser som finns och i loopen anropas Dice.roll() i varje Dice instans som finns för att slumpa fram ett nytt värde för varje enskild Dice instans. Sedan i loopen läggs det till nya klasser till elementen i graphicDiceArray så att grafiken på en tärning uppdateras för att motsvara vilket värde varje enskild tärning fick. Efter detta anropas displayScore() och audio.play().
- **createDOM()**: Detta är en metod som ansvarar för att skapa alla DOM element för applikationen och applicerar händelselyssnare på vissa av DOM elementen.
- **mouseDown():** Denna metoden används för att starta en drag and drop process av ett applikationsfönster. Denna metoden används för att sätta dragStartX och dragStartY till den position användaren har placerat på. Den används dessutom för att anropa DiceApp.adjustZIndex() metoden för att plasera applikationsfönstret över alla andra applikationsfönster då det nu är det "aktiva" fönstret. Dessutom sätter metoden två nya eventlyssnare till dokumentet, mousemove och mouseup som anropar metoderna "mouseMove" och "mouseUp"
- **mouseMove():** metod som uppdaterar var applikationsfönstret plaseras grafiskt på gränssnittet samt uppdaterar dragStartX och dragStartY för fönstrets nya position då man drar i fönstret.
- **mouseUp():** metod som tar bort eventlyssnare för mouseMove och mouseUp då man släpper ett fönster så att det "droppas" på den positionen användaren har flyttat det till.
- **adjustZIndex():** Sätter Z indexet av det aktiva fönstret till att motsvara tiden användaren aktiverade fönstret genom att skapa ett "new Date()" och anropa "getTime()" metoden på Date instansen. Eftersom getTime alltid kommer returnera ett värde större än föregående getTime anropas så kommer det aktiva fönstret bli placerat överst.
- **displayScore()**: Metoden har i uppgift att läsa av det totala värdet av alla sammanlagda tärningars värde. Om poängen är mer eller samma som 0 så skapas en array "numClass" med strängarna "one","two" [...] "nine" som alla är klassnamn som ska appliceras på fem <li> element som är children till ulCounter som var och en ska representera en enskild siffra. För att t.ex. Representera talet "11" så måste de tre första <li> elementen ha klassnamnet "zero" och de två sista "one". Det innebär att talet 11 måste delas upp i ett tiotal och ett ental dvs, 1 som tiotal och 1 som en tal. Detta görs genom att använda metoden .toString() på score variabeln som håller diceOnBoardValues värde. .toString() returnerar enligt Mozilla (2022) en

sträng innehållande samma värde som det som .toString() metoden applicerades på. .split() metoden används sen på det värdet som returnerats från .toString() metoden. .split() tar emot parametern (“”) som är en tom sträng, vilket innebär att strängen ska delas upp så att varje enskild karaktär i strängen hamnar i sitt egna index i en array. Strängen “11” skulle då blir array = [“1”, “1”] där index 0 representerar tiotalet i diceOnBoardValue och index 1 representerar entalet. När arrayen skapats körs en .map() metod. Enligt Mozilla (2022) så returnerar .map() metoden en ny array efter att en definierad funktion utförts på varje element i arrayen. .map() metoden tar emot parametern “Number” vilket innebär att för varje element i arrayen .map() utförs på så kommer varje element omvanldas till en siffra istället för en sträng. På en rad kod utförs tre seperata metoder på ett tal för att dela upp talet i exempelvis 1000tal, 100tal, 10tal och ental. Det som returneras efter denna operation är en array innehållande de olika enskilda delarna av talet och lagras i variabeln scoreArray. Metoden .reverse() används sedan på scoreArray. Detta då siffran delas upp till tiondelar, hundradelar osv så måste ordningen omvandlas så att entalen alltid är på scoreArray[0], 10talen blir alltid index 1, 100tal index 2 osv… efter att .reverse() metoden utförts, får det fjärde <li> elementet (det element som grafiskt är längst till höger på poängräknaren) klassnamnet “numClass[scoreArray[0]]”. Detta innebär att om t.ex. Entalet är siffran 3 så represnterar scoreArray[0] värdet 3. Index 3 på numClass representerar strängen “three” vilket innebär att det första <li> elementen som representerar heltalet får klassnamnet “three”. Denna logiken gör så att poängräknaren fylls på från höger till vänster.

- **deleteObjs()**: Metoden ansvarar för att ta bort en DiceApp instans från DOMen. Först används en this.windowWrapper.remove() metod som enligt Mozilla (2022) tar bort ett element från DOMen. Efter detta sker en for-in loop som stegar igenom alla egenskaper, inklusive de egenskaper som dyker upp i ett objektets prototyp (Stefanov, s.160). Denna for-in loopen utförs på “this” nyckelordet i kontexten av en DiceApp instans vilket innebär att aktuell DiceApp instans egensakper kommer att tas bort i for-in loopen genom delete operatorn. Delete operatören kommer enligt Mozilla (2022) ta bort en specifikt egenskap från ett objekt.

Dice klassen: Dice klassen ansvarar för att hantera logiken för tärningarna. Klassen har en egenskap:

- **value**: Egenskapen håller värdet av det senaste tärningskastet från Dice.add() metoden. Egenskapen används i DiceApp för att hämta värdet av det senaste tärningskastet.

Klassen har två metoder:

- **roll():** Slumpar ett värde mellan 1-6 och uppdaterar Dice instansens “value” egenskap
- **destroy():** Metoden ansvarar för att ta bort en Dice instans från DOMen. En for-in loop används som stegar igenom alla egenskaper, inklusive de egenskaper som dyker upp i ett objekts prototyp (Stefanov, s.160). Denna for-in loopen utförs på “this” nyckelordet i kontexten av en Dice instans vilket innebär att aktuell Dice instans egensakper kommer att tas bort i for-in loopen genom delete operatorn. Delete operatören kommer enligt Mozilla (2022) ta bort en specifikt egenskap från ett objekt.

Referenser:

Rollbar.com. 2022. *Javascript constructors*.
https://rollbar.com/blog/javascript-constructors/# (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *DataTransfer*.
https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. Document.*getElementsByClassName()*. https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *Audio()*. https://developer.mozilla.org/en-US/docs/Web/API/HTMLAudioElement/Audio (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *Array.prototpye.splice()*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *Array.prototpye.from()*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/from (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *Array.prototpye.map()*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *Object.prototpye.toString()*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *Delete operator*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete (Hämtad 10 August 2022).

Developer.mozilla.org. 2022. *Element.remove()*. https://developer.mozilla.org/en-US/docs/Web/API/Element/remove (Hämtad 10 August 2022).

Developer-mozilla.org. 2023. *function.prototype.bimd()*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind (hämtad 1 jun 2023)

Stefanov, Stoyan. 2013. *Object-Oriented JavaScript.* 2. uppl. Birmingham: Packt Publishing Ltd.