Endabgabe EIA1

(Nur diejenigen, welche keine volle rote Ampelstufe auf der Kursseite haben dürfen hier teilnehmen. Abgaben von Leuten welche es nicht soweit geschafft haben werden ignoriert. Diese müssen den praktischen Teil nächstes Semester wiederholen.)

[AUFGABE]

Programmieren sie ein simples Kartenspiel.

Konzept ist nicht gefordert, aber empfohlen.

Finales Abgabedatum: SO, 28.07.2019, 23:59

[ERWARTETER PROGRAMMABLAUF]

Spielregeln des Kartenspiels: Jeder Spieler kann pro Runde eine Karte spielen, welche entweder die gleiche Farbe oder die gleiche Wertigkeit der letzten gespielten Karte besitzt. Wenn man nicht spielen kann muss man stattdessen eine Karte ziehen. Der erste Spieler ohne Karten gewinnt.

Jede Farbe besitzt von jeder Wertigkeit eine Karte, alle Karten sind einzigartig und kommen entsprechend nicht doppelt vor.

Der Spieler spielt gegen einen sehr einfachen Computer. Zu Beginn des Spiels erhalten beide Seiten eine Anzahl von Handkarten vom Kartenstapel. (mindestens 3, Anzahl ist euch überlassen)
Eine Karte wird aus dem Kartenstapel genommen und als Startkarte auf den Ablagestapel gesetzt.

Der Spieler kann dann auf eine seiner Handkarten klicken, woraufhin diese dann gespielt wird (insofern regeltechnisch erlaubt). Wenn er eine Karte spielen konnte ist der Gegner danach dran.
Sollte jedoch keine Handkarte passen, muss der Spieler stattdesssen auf den Kartenstapel klicken, um eine weitere Karte zu ziehen und den Gegner den Zug zu überlassen.
// Ihr könnt auch einen weiteren Spielzug nach dem Kartenziehen erlauben, Variation ist hier sehr wohl möglich.

Der Gegner versucht immer eine Karte zu spielen. Dies darf die nächstbeste Karte sein. Falls keine Karte spielbar ist muss der Computer eine Karte ziehen. Danach ist der Spieler wieder an der Reihe.
// Oder der Computer muss auch solange Karten ziehen, bis er eine spielen kann.

Das Spiel benötigt keine spezielle Animation bei dem spielen oder ziehen einer Karte. Diese kann einfach sofort-artig auf dem Ablagestapel erscheinen.

[BENOTUNG]

Note der End-Abgabe basiert auf den dafür ausgefüllten Anforderungs-Punkten.
Bis zu ein Punkt darf dabei nicht erfüllt sein um die 1 zu erhalten.
Jeder weitere fehlende Punkt verringert die Note um 1. (4.0 >= Bestanden)
Jedes erfüllte Optionale Ziel erhöht eure Note um 1. (Maximale Bestnote: 1).

Subjektiv für die Betreuer, für nicht ausreichendes Erfüllen eines Punktes wird dieser gegenfalls nicht gegeben.
Ebenfalls wird für objektiv schlechte Farbwahl und Ausführung ("Augenkrebs", fiese Kontrastfarben, enorm viele Rechtschreibfehler) eine Note abgezogen.

[TIPPs]

Macht euch ein Konzept!
Versucht den Spielfluss dieses Kartenspiels für nur eine Person in einem Flowchart einzufangen.
An welchen Punkten werden Entscheidungen gefällt? Muss der Computer gewisse Sachen abfragen?
Wann muss der Spieler interagieren, wie interagiert er, was sind seine Optionen?
Welchen Ablauf hat der Computer, wie wählt er seine Karte aus?
Wie sehen Karten aus? Welche Elemente benötigen beim Erstellen einen Eventlistenener?

Macht euch darüber Gedanken, wann die aktuelle Spielfläche mit dem vorhandenen Karten neu dargestellt werden muss!

[ZU ERFÜLLENDE ANFORDERUNGS-PUNKTE]

Gesamt: 25.
(25 = 1.0, 24 = 1.0, 23 = 2.0, 22 = 3.0, 21 = 4.0, <= 20 = NB)
Einfache davon: 10. (Sollten in unter einem Tag machbar)
Optionale: 5.
Deine Endabgabe soll dabei folgendes erfüllen:

[HTML]

1. [ ] Einen einfachen, grauen Header mit deinem Namen und Matrikelnummer.
2. [ ] Semantische HTML-Tags wurden genutzt, Tags werden sinnvoll eingesetzt.

Empfehlenswerter Aufbau für das HTML:
Vier generelle Bereiche:
Der Gegner besitzt eine Fläche, in welche seine (verdeckten) Handkarten dargstellt werden können.
Es gibt einen Stapel zum ziehen von Karten.
Es gibt eine Spielfläche, auf welcher die gespielten Karten erscheinen werden.

Eine Fläche für die eigenen Handkarten.

[CSS]

3. [ ] Die Spielflächen besitzt eine passende Hintergrundfarbe (oder Hintergrundtextur). Kein Augenkrebs!
4. [ ] Spielflächen für Gegner und den Spieler verändern ihre Größe im Spielverlauf nicht und sind vom restlichen Feldern klar sichtbar getrennt (entweder über sichtbare Border oder andere Hintergrundfarbe).

[CSS-Spielkarten]

5. [ ] Die Spielkarten haben eine einheitliche Größe.
6. [ ] Die verschiedenen Kartenfarben werden über Klasse definiert.

Ihr könnt auch Bilder nutzen.

TIPP: Vielleicht wäre es sinnvoll, erst eine einzelne Karte im HTML und CSS zu designen. Den Aufbau/die Klassen dieser Karte können später über eine Generierungsfunktion im Skript nachempfunden werden.

TIPP: Mittels CSS lässt sich ziemlich einfach eine Spielkarte designen. Abgerundete Ecken sind sehr einfach umzusetzen, und mittels Rotation kann man das umgedrehte Kartensymbol auf der unteren Seite sehr einfach gestalten.

[Coding]
[Grundlegend]

7. [ ] Euer Skript kann ohne Fehler aus dem JS benutzt werden.
8. [ ] Euer Skript generiert mindestens 32 Karten. (8 Wertigkeiten X 4 Farben). Jede Karte kommt nur ein mal vor. Die Karten-Generierung kann Hardcoded sein. Empfehlung ist eine verschachtelte For()-Schleife.
9. [ ] Karten-Wertikgeit könnt ihr nach eigener Wahl treffen. Wenn eure Karten Sonderregeln besitzen, so muss eine Erklärung im HTML sein. Es müssen mindestens 8 verschiedene Wertigkeiten besitzen.
Beispiel für Wertigkeiten: Zahlen von 0-9, 10, Bube, Dame, König, Ass. Oder erfindet selbst welche.
10. [ ] Karten können dabei ebenfalls die folgenden "Farben" annehmen: Herz, Karo, Kreuz, Pik - oder auch einfache Farben wie: Rot, Blau, Grün, Gelb.
11. [ ] Euer Skript besitzt mehrere Arrays. Eines für den Karten-Ziehstapel, eines für die Spieler-Hand, eine für die Gegner-Hand und eines für den Karten-Ablagestapel.

12. [ ] Euer Karten-Array kann mittels einer Funktion gemischt werden. Eigenrecherche möglicherweise Notwendig. Zu Beginn des Spiels werden die Karten gemischt.
13. [ ] Karten werden zwischen den verschiedenen Arrays mittels der Push()- und Splice()-Funktion verschoben. Wenn eine Karte gespielt wird, wird sie erst

in das neue Array gepusht, danach aus ihrem Ursprungsarray gelöscht.
14. [ ] Baut euch Funktionen, mit welchen der Spieler oder der Computer eine Karte vom Stapel ziehen kann.
15. [ ] Baut eine Funktion, mit welcher die i-te Karte auf der Hand gespielt werden kann. Diese Funktion muss mittels EventListener "click" auf jede HTML-Karte des menschlichen Spielers gelegt werden.

TIPP: Ihr werdet noch weitere Funktion benötigen außer diesen hier. Der Computer muss ebenfalls in der Lage sein, Karten zu ziehen.

[HTML-Generierung]

16. [ ] Euer Skript besitzt eine Funktion um eine aufgedeckte Karte aus HTML-Elementen zu erstellen.
17. [ ] Euer Skript besitzt eine Funktion um eine verdeckte Karte aus HTML-Elementen zu erstellen.

TIPP: Lasst alle Karten zu Beginn offen verteilen. Wenn ihr eueren Code weit genug geschrieben habt könnt ihr euch um verdeckte Karten kümmern. Dies macht es euch einfacher zu kontrollieren ob alles soweit funktioniert. Alternative Lösungen, wie: jeder Karte besitzt eine HTML-Element, welches die Karte verdecken kann, werden auch akzeptiert für diese beiden Punkte.

[Computer-Gegner]

18. [ ] Der Computer-Gegner versucht nach dem Zug des Spielers automatisch eine eigene Karte auszuspielen.
19. [ ] Der Computer versucht dabei nicht "intelligent" den Spieler zu besiegen, sondern wählt die nächst-beste legbare Karte aus - oder ihr baut einen schlauen Algorithmus.
20. [ ] Konnte der Computer in seinem Zug keine Karte spielen, so muss er vom Kartenstapel ziehen.

TIPP: Findet einen Weg, um die oberste Karte auf dem Ablage-Stapel anzuschauen und mit einer Karte auf einer Hand zu vergleichen. Möglicherweise ist eine Funktion hilfreich, welche überprüft ob eine Karte auf den aktuellen Stapel ablegbar ist, und entsprechend WAHR oder FALSCH zurückgibt. Eine solche Funktion könnte sowohl beim menschlichen Spieler als auch Computer praktisch sein.

[Mensch-Spieler]

21. [ ] Karten ausspielen: Vergleicht eure auszuspielende Karte mit der obersten Karte des Ablage-Stapels.
Verglichen wird dabei die Farbe der Karte und die Wertigkeit. Karten können nur dann gespielt werden wenn sie die gleiche Farbe haben ODER eine gleiche Wertigkeit besitzen.
Sollte diese Bedingungen erfüllt sein, so wird diese Karte auf den entsprechenden Stapel gelegt und der andere Spieler ist am Zug.

Kann durch Klick des Spielers auf eine seiner eigenen Karten ausgelöst werden oder automatisch durch den Computer, wenn eine Karte passen sollte.

[Interface]

22. [ ] Ein Karten-Interface wurde definiert. Jede Karte weiß, welche Wertigkeit und Farbe sie hat.
23. [ ] Um die Karten darzustellen werden HTML-Elemente generiert und passend zusammengesetzt.
24. [ ] Das sichtbare HTML orientiert sich an dem im Hintergrund laufendem Javascript. Jede auf dem Spielfeld sichtbare Karte spiegelt eine im Skript erstellte Instanz des Karten-Interfaces wieder.

Das Interface darf auch mehr Variablen besitzen. Dieser werden nicht bewertet.

[Spieler-Interaktion]

25. [ ] In seinem eigenen Zug darf der Spieler ebenfalls nur Karten spielen, welche auf die aktuelle Karte des Ablage-Stapels passen würde, oder alternativ dazu auf den Karten-Ziehstapel klicken, wodurch diese Karte dann auf die Spieler-Hand wandert.

[OPTIONAL]

Optionale Ziele MÜSSEN nicht erfüllt werden! Sie fließen jedoch positiv in die Berwertung mit ein. Manche Punkte der optionalen Ziele werden Subjektiv bewertet.

OPT1. [ ] Gewisse Wertigkeiten eurer Karten besitzen Sonderregeln. Beispielsweise: Gegner muss bei einer 7 zwei Karten ziehen.
OPT2. [ ] Schönes, herausgearbeitetes CSS, ohne Augenkrebs.
OPT3. [ ] Das Gewinnen des Spiels erhält der Spieler eine passende Nachricht.
OPT4. [ ] Gegner-KI ist etwas ausgefeilter und plant vorraus, schaut sich die Spieler-Handkarten an und agiert entsprechend.
OPT5. [ ] Sollte der Ziehstapel leer sein wird dieser aus dem Ablagestapel neu zusammengemischt.

[NEBENSÄCHLICHES]

[UPLOAD]

Abgabe muss im Steckbrief verlinkt sein.

Für diejenigen, welche ihre Abgabe nicht direkt sichtbar ins Web hochladen wollen, gibt es die Option eine .zip hochzuladen.
Dann muss die .zip im Steckbrief verlinkt sein.

Abgabe muss eine .html-Datei, eine .css-Datei, eine .js-Datei und eine .ts-Datei enthalten. Typescript soll dabei die Javascript-Datei "widerspiegeln".

[VERSPÄTUNG/NICHTABGABE]

Nicht gefundene Abgaben werden als nicht abgegeben bewertet. Abgaben, welche nach der Frist hochgeladen/verlinkt wurden, werden NICHT kontrolliert! Es gibt keine Verlängerung dieser Abgabe.
(Solltet ihr "Ferien" bzw. vorlesungsfreie Zeit machen wollen ist das nicht mein Problem. Zu spät anzufangen ist keine Entschuldigung).

[KOPIEREN]

Benutzt kein urheberrechtlich geschütztes Material.
Sollte ein Plagiatsversuch gefunden werden, wird entsprechend gehandelt.

[Kommentare]

Benutzt Kommentare im Code, falls möglich. Sollten wir einen Plagiatsfall entdecken, so sind Kommentare praktisch um zu beweisen dass man den Code selber geschrieben hat.