

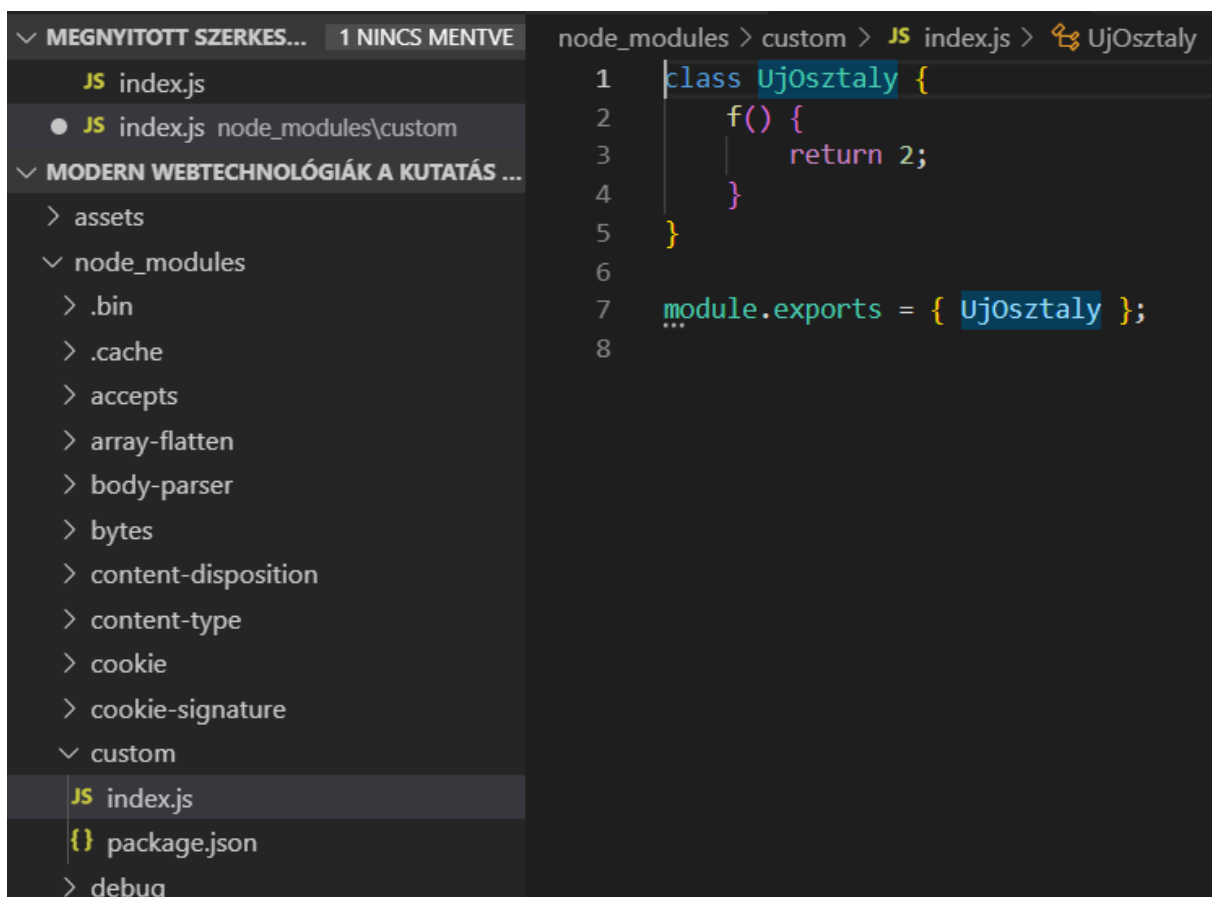
Modern webtechnológiák a kutatás szolgálatában II.

Miről is volt szó az órán?

Az órán beszéltünk **node** szerverekről, példát néztünk **expressJs** szerverre illetve **VueJs**-el készítettünk frontendet, és beadandóként abból egy feladatot.

Backend rész:

- Beimportáltunk egy Stacket (spque) ennek vizsgáltuk a működését.
- Majd létrehoztunk egy új npm modult a node_modulesban, ami nálam **custom** nevet viselt és ezt példányosítottuk.
- Ez a custom osztály csak egy metódust tartalmazott ami visszaadott egy számot (2) de itt a modul exportálás és a fő index.js-be importáláson volt a hangsúly.



```
node_modules > custom > JS index.js > UjOsztaly

1  class UjOsztaly {
2      f() {
3          return 2;
4      }
5  }
6
7  module.exports = { UjOsztaly };
8
```

- Ezután jött az expressJs-es rész, ahol a root-ra és a /cica route-ra küldtünk egy üzenetet.

- A következő oldalon lévő fénykép tartalmazza a Stack-es példát és az expressJs-est.

```
JS index.js > ...
1  var Stack = require('spque').Stack;
2  let q = new Stack();
3  q.put(2);
4  q.put(4);
5  console.log(q.size, q.get());
6
7  const UjOsztaly = require('custom').UjOsztaly;
8  let ujOsztaly = new UjOsztaly();
9  console.log(ujOsztaly.f());
10
11 const express = require('express');
12 const app = express();
13
14 app.get('/', (req, res) => { res.send('Balhél :)'); });
15 app.get('/cica', (req, res) => { res.send('Cica Balhél :)'); });
16
17 app.listen(3000, () => console.log('App listening on port: 3000!'));
18
```

- Ezzel le is zárult a backend rész, a következőkben a frontenddel fogunk foglalkozni.

Frontend: Elég érdekes volt számomra mivel Angularban és Ionicban fejleszték.

Amik érdekes voltak és megkellett szokni:

- Megkellett szokni a sok v-s és :-os előtagot a html tagekben.
- Azt, hogy hogy minden egy fájlban van.
- Hogy vannak kitüntetett adattagok és metódusok a script részben:
 - data()
 - methods
 - beforeMount
- Ezek mint újak voltak számomra amik lassították a fejlesztési folyamatot, bár látszik is, hogy angularból loptam számomra megszokott dolgokat (pl.: onInit() metódus)

Térjünk rá a kódra, először is a játékról szeretnék beszélni mivel azt is lehetett készíteni, én egy kezdetleges **Chicken Hunter** játékot csináltam ahol későbbi verzióban lehetne több pálya illetve mozgás, jelenleg úgy oldottam meg a játékelményt, hogy változó sebességben jelennek meg csirkék a pályán, amiből ha túl sok lesz veszítettünk, illetve ha adott mennyiségű score-t gyűjtünk akkor gyorsul a játék (level up).

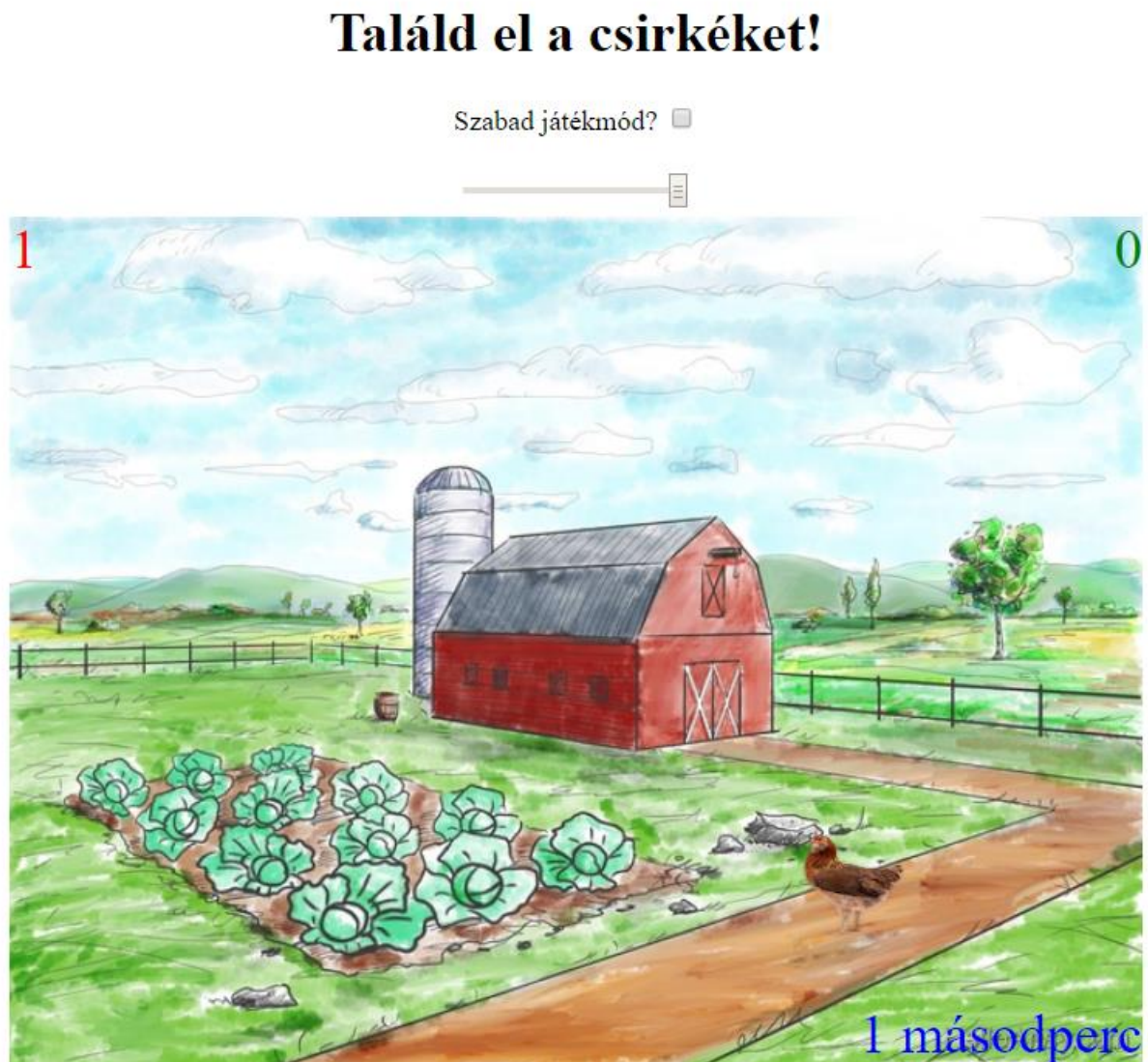
Megjegyezném, hogy először rosszul kezdtem el használni a vue-t, mivel teljesen ismeretlen fogtam fel (multimédia óráról használatos, document.getElementById() és ehhez hasonló pure javascript dolgokkal próbáltam manipulálni a játékot, amivel nem is sikerült elérni azt amit szerettem volna, mivel nem működött jól együtt a vue-val, ezért volt olyan rész mikor a játék kezdetleges logikáját teljesen újra kellett írnom.

```
chicken_hunter.vue X
chicken_hunter.vue > {} "chicken_hunter.vue" > script
1 <template>
2 <div class="center">
3   <label>
4     <h1>Találd el a csirkéket!</h1>
5   </label>
6   <label>
7     Szabad játékmód?
8     <input type="checkbox" v-model="isFreeplay" />
9   </label>
10  <br />
11  <br />
12  <input
13    type="range"
14    :min="maxSpeed"
15    :max="minSpeed"
16    v-model="speed"
17    :disabled="isFreeplay == false"
18  />
19  <br />
20
21  <label v-if="levelUpCondition == true">{{ levelUpMessage }}</label>
22  <div id="playground" class="playground">
23    <label class="chicken-counter">{{ chickenCounter }}</label>
24    <label class="point-counter">{{ counter }}</label>
25    
34    <label class="speed">{{ speed/1000 }} {{speedUnit}}</label>
35  </div>
36 </div>
37 </template>
```

Na de térjünk vissza a kódhoz, a template rész, ami a page megjelenését tartalmazza HTML-ben íródott, de több sajátos vue tulajdonságot is visel amit már említettem, ezekről majd screenshotokkal fogok beszámolni, elég rövidre sikerült mivel minden fontosabb dolgot a logika old meg.

Ugye mivel vue-ról van szó, a template csak egy div-et tartalmaz, és minden benne található, ez így van most is, amit internal CSS-el oldottam meg alul a style tagben, ahol a page css része található, erről majd később.

A játék alap kinézete:



A játék fő mondanivalója nem más mint „Találd el a csirkéket!”, egy h1-es label, ezután közvetlenül van téve egy kérdés, hogy „Szabad játékmód?” Ez annyit takar, hogy ha a játékot nem akar szabályokat, akkor bekapcsolja és maga szabályozhatja a csirkék megjelenési idejét(spawn), nem lesz vége a játéknak és nem tud kikapni (nagyon sok csirke lesz a pályán

A **playground** bal felső sarkában egy piros számmal van jelezve, hogy hány csirke található a pályán akit le kellene lőni, a jobbfelső sarokban pedig zölddel, hogy már mennyit sikerült eltalálni az aktuális játékmenet alatt.

A jobb alsó sarokban az éppen aktuális spawn időt láthatjuk kék színnel, másodpercben. Ez dinamikusan változik ahogy szintet lépünk vagy szabad játékmódban módosítjuk a spawn időt a csúszka segítségével.



Ha túl sok csirke lett a pályán akkor veszünk, ez a szám konstansként van megadva a játék elején, mint ahogyan sok már is, így könnyen módosítható. Ez a szám most 10.

Találd el a csirkéket!



Ezennel meg is ragadom az alkalmat és a konstansokat tartalmazó screenshottal elmondom, hogy mi mi célt szolgál:

```
39  <script>
40  const ENEMY_WIDTH = 60;
41  const ENEMY_HEIGHT = 60;
42  const MAP_WIDTH = 650;
43  const MAP_HEIGHT = 488;
44  const ENEMIES_LIMIT = 10;
45  const MIN_SPEED = 3000;
46  const MAX_SPEED = 500;
47  const START_SPEED = MIN_SPEED;
48  const SPEED_MODIFICATION = 500;
49  const LEVEL_UP_LIMIT = 20;
50  const LEVEL_UP_TIMEOUT = 2000;
51
52  const MOTIVATION = "Találd el a csirkéket!";
53  const FREEPLAY = "Szabad játékmód?";
54  const SPEED_UNIT = "másodperc"
55
56  const LEVEL_UP_MESSAGE = "SZINT LÉPÉS, A JÁTÉK GYORSULT!";
57  const GAME_FINISHED =
58  | "GRATULÁLOK, SIKERÜLT VISSZAVERNI A CSIRKÉK TÁMADÁSÁT. A JÁTÉK VÉGET ÉRT!";
59  const GAME_OVER = "VESZTETTÉL, A CSIRKÉK LEGYŐZTEK. A JÁTÉK VÉGET ÉRT!";
60
```

- ENEMY_WIDTH = A csirke képének szélessége (ugyanez a szám van megadva css-ben is (number).
- ENEMY_HEIGHT = A csirke képének magassága (ugyanez a szám van megadva css-ben is (number).
- MAP_WIDTH = A pálya szélessége (ugyanez a szám van megadva css-ben is a background-nak (number).
- MAP_HEIGHT = A pálya szélessége (ugyanez a szám van megadva css-ben is a background-nak (number).
- ENEMIES_LIMIT = Maximálisan ennyi csirke lehet egyszerre a pályán (number).
- MIN_SPEED = A játék leglassabb sebessége, a csúszka maximuma (number).
- MAX_SPEED = A játék leggyorsabb sebessége, a csúszka minimuma (number).
- START_SPEED = A játék kezdő sebessége (number).
- SPEED_MODIFICATION = Szintlépéskor ennyit gyorsul a játék (number).
- LEVEL_UP_LIMIT = Ennyi score-ként lép szintet a játékos (gyorsul a játék) (number).
- SPEED_UNIT = Az idő kijelzésére használt szöveg (string).
- MOTIVATION = A „Találd el a csirkéket!” szöveg (string).
- FREEPLAY = a „Szabad játékmód?” kérdésre használt szöveg (string).
- LEVEL_UP_MESSAGE = A szintlépéskor használt szöveg (string).
- GAME_FINISHED = Ha megnyertük a játékot ez a gratuláló szöveg vár minket (string).
- GAME_OVER = Ha elvesztettük a játékot ez a szöveg vár minket (string).

Betöltjük az assets mappából a csirkéket tartalmazó képeket egy tömbbe, 3 különbözőt töltöttem be és ezek közt random szám generáló segítségével fogok választani minden egyes spawn-nál a tömbből.

Ezeket a konstansokat muszáj értékül adnunk a komponensben található változóknak.

Ezt a data() metódusban tesszük meg:

```
61 export default {
62   data() {
63     return {
64       enemiesLoaded: [
65         require("./assets/chicken1.png"),
66         require("./assets/chicken2.png"),
67         require("./assets/chicken3.png")
68       ],
69       enemies: [],
70       counter: 0,
71       speed: START_SPEED,
72       minSpeed: MIN_SPEED,
73       maxSpeed: MAX_SPEED,
74       speedUnit: SPEED_UNIT,
75       isFreeplay: false,
76       levelUpCondition: false,
77       levelUpMessage: LEVEL_UP_MESSAGE,
78       chickenCounter: 0,
79       motivation: MOTIVATION,
80       freeplay: FREEPLAY
81     };
82   },
83 }
```

A játékban használt metódusok:

- `clickListener(i: number)`: Ha csirkére kattintunk ez a metódus fut le, ellenőrzi a játék pontszámait és az alapján cselekedik.
- `getRndInteger(min: number, max: number)`: Véletlenszerű szám generálására használt metódus.
- `loop()`: A játék futtatását megvalósító rekurzív függvény.
- `onInit()`: `beforeMount`-ban meghívott játékot inicializáló függvény.
- `onGameOver()`: Ha vége van a játéknak ez a metódus fut le.

Most nézzük meg ezeket a metódusokat részletesebben:

`clickListener(i)`:

```
83     clickListener(i) {  
84         if (this.chickenCounter >= ENEMIES_LIMIT && !this.isFreeplay) {  
85             return;  
86         }  
87         this.$set(this.enemies[i], "class", "enemy enemy-death");  
88         this.counter++;  
89         this.chickenCounter--;  
90     },
```

Vue jellegzetességéből adódóan egy `img` HTML taget rak ki egy ciklusban ami egy tömbből tölti be a csirkéket. Pontosán ezért ismert minden csirke fénykép tömb indexe, ezt használjuk itt `i`-ként, mint index és mint a `for` ciklus ciklusszámlálója. Ha a pályán lévő csirkék aktuális száma nagyobb vagy egyenlő a maximálisan megengedettnél (konstans) és ha nem szabad játékmódban vagyunk akkor véget is ér a click event, `return`-öl a metódus, viszont ha nem akkor az aktuális csirke css class-ját módosítjuk a „halált” animáló osztályra, növeljük score szám válzotóját (`counter`) és a csirke számlálására használt változót pedig csökkentjük (`chickenCounter`).

`getRndInteger(min, max)`:

```
91     getRndInteger(min, max) {  
92         return Math.floor(Math.random() * (max - min)) + min;  
93     },
```

Erről nem hiszem, hogy sokat kell beszélni, alap JavaScript matematikus függvénytár metódusa, ami úgy van paraméterezve, hogy egyszerűen meglehessen neki mondani, hogy minimum és maximum mekkora szám lehet generálva.


```
95 loop() {
96   if (this.counter >= LEVEL_UP_LIMIT && !this.isFreeplay) {
97     if (this.speed == MIN_SPEED) {
98       this.onGameOver();
99       this.levelUpCondition = true;
100       this.levelUpMessage = GAME_FINISHED;
101       return;
102     }
103     this.speed -= SPEED_MODIFICATION;
104     this.counter = 0;
105     this.levelUpCondition = true;
106     setTimeout(() => {
107       this.levelUpCondition = false;
108     }, LEVEL_UP_TIMEOUT);
109   }
110   const randForImage = this.getRndInteger(0, this.enemiesLoaded.length);
111   const randForTop =
112     this.getRndInteger(
113       ENEMY_WIDTH / 2,
114       MAP_HEIGHT - ENEMY_HEIGHT
115     ).toString() + "px";
116   const randForLeft =
117     this.getRndInteger(
118       ENEMY_WIDTH / 2,
119       MAP_WIDTH - ENEMY_WIDTH
120     ).toString() + "px";
121
122   this.enemies.push({
123     src: this.enemiesLoaded[randForImage],
124     id: "enemy",
125     class: "enemy enemy-spawn",
126     style: {
127       top: randForTop,
128       left: randForLeft
129     }
130   });
131   this.chickenCounter++;
132   if (this.chickenCounter >= ENEMIES_LIMIT && !this.isFreeplay) {
133     this.onGameOver();
134     this.levelUpCondition = true;
135     this.levelUpMessage = GAME_OVER;
136     return;
137   }
138   window.setTimeout(this.loop, this.speed);
139 },
```

Ez egy picit hosszú kép lett, de itt megpróbálom röviden leírni mit csinál ez a metódus, amíg fut a játék mindig meghívja magát az aktuális játék sebességével timeoutolva, azelőtt. Na de kezdjük a legelejétől.

Ha a elértem annyi pontot, hogy szintet lépjek és nem szabad játékban vagyok akkor szintet lépek, kivéve ha már a leggyorsabb játékban vagyok (utolsó szint) mert ha ez megtörténik akkor vége a játéknak, és gratuláló üzenetet kapok. Viszont, ha nem ez

történik meg akkor szintet lépek normális esetben, a sebesség növekedik, a csirkék számolására használt változó nullázódik és egy timeoutot vezérlő boolean változó is értéket kap, ami 2 másodperc múlva false-t kap, ezért véget ér a szint lépés szöveg megjelenése a page-n.

Ezután random számokat generálok, hogy melyik csirkét vegyem ki a tömbből és hogy hol helyezzem el az új csirkét a pályán. Ha összeállt az objektum akkor belerakom a tömbbe, növelem a csirkék számát tartalmazó változót majd ezeket után ellenőrzöm, hogy vége lett-e a játéknak, tehát, hogy a csirkék száma elérte-e a maximumot és hogy nem szabad játékmódban vagyunk. Ha igazra fut ez a feltétel, akkor vége lesz a játéknak, és a tájékoztat arról a játék, hogy vesztettünk. Ez után return-öl a metódus, hogy ne tudjon újabb frame-t nyitni, tehát megszakad a rekurzív metódus. Viszont, ha ez nem teljesül akkor egy setTimeout-tal újra meghívja a loop metódust bizonyos idő múlva (speed ami a játék sebességét tárolja).

onInit, onGameOver és a beforeMount metódus:

- Az onInit metódusban hívjuk meg a loop() metódust, ekkor indul el a játék.
- onGameOver metódusban kiürítjük a csirkéket tartalmazó tömböt (amiből jönnek a csirke fényképek) és nullázuk a számlálókat.
- beforeMount metódusban hívjuk meg az onInit metódust. Igen, itt látni egy felesleges kört, de mint említettem direkt van így nekem az Angularból megszokva, onInit-ben sok egyéb más dolgot is lehetne csinálni ami nem lehet része a loop metódusnak s úgymond erre van felkészítve ez a folyamat.

```
141   onInit() {  
142     this.loop();  
143   },  
144   onGameOver() {  
145     this.enemies = [];  
146     this.counter = 0;  
147     this.chickenCounter = 0;  
148   }  
149 },  
150 beforeMount() {  
151   this.onInit();  
152 }  
153 };  
154 </script>
```

Ezzel véget ért a kód logikája, mehetünk a css részre, viszont ezt már tényleg röviden fogom részletezni, mivel ez nem része a vue-nak, ez HTML és nem része az óra tárgyának.

A CSS rész: A **playground** osztály a játék játékterét jelenti, a background image méretét veszi fel és megkapja a képet.

A **center** osztály középre igazítást végez csak.

Az **enemy** osztály a csirke fényképén lesz majd, méretet és pozíciót állít.

Az **enemy-spawn** osztály animációt állít ami a csirke megjelenésekor fut le.

Hasonlóképp az **enemy-death**-el, csak ez a csirkére kattintás után fut le.

A **chicken-counter** osztály a playground bal felső sarkában található pontszám kijelzését valósítja meg.

Hasonlóképp a **point-counter**-rel, ami a pontszámot jeleníti meg a playground jobb felső sarkában.

```
156 <style>
157 .playground {
158     background-image: url("../assets/background.jpg");
159     width: 650px;
160     height: 488px;
161     margin: auto;
162     position: relative;
163 }
164 .center {
165     text-align: center;
166 }
167 .enemy {
168     width: 60px;
169     height: 60px;
170     position: absolute;
171 }
172 .enemy-spawn {
173     animation-name: appearAnimation;
174     animation-duration: 250ms;
175     animation-fill-mode: forwards;
176 }
177 .enemy-death {
178     animation-name: disappearAnimation;
179     animation-duration: 1000ms;
180     animation-fill-mode: forwards;
181 }
182 .chicken-counter {
183     color: ■ red;
184     font-size: 200%;
185     position: absolute;
186     top: 0;
187     left: 0;
188 }
189 .point-counter {
190     color: ■ green;
191     font-size: 200%;
192     position: absolute;
193     top: 0;
194     right: 0;
195 }
196 .speed {
197     color: ■ blue;
198     font-size: 200%;
199     position: absolute;
200     bottom: 0;
201     right: 0;
202 }
203
```

A **speed** osztály pedig a playground jobb alsó sarkában az időkijelzést.

Ezentúl még vannak az animációk amik az utolsó képként jelennek meg a dokumentációban.

```
203 @keyframes appearAnimation {
204   0% {
205     opacity: 0;
206     height: 0px;
207     width: 0px;
208   }
209   25% {
210     opacity: 0.25;
211     height: 15px;
212     width: 15px;
213   }
214   50% {
215     opacity: 0.5;
216     height: 30px;
217     width: 30px;
218   }
219   75% {
220     opacity: 0.75;
221     height: 45px;
222     width: 45px;
223   }
224   100% {
225     opacity: 1;
226     height: 60px;
227     width: 60px;
228   }
229 }
230
231 @keyframes disappearAnimation {
232   0% {
233     opacity: 1;
234     transform: rotateX(90deg);
235   }
236   50% {
237     opacity: 0.5;
238     transform: rotateX(0deg);
239   }
240   100% {
241     display: none;
242     opacity: 0;
243     transform: rotateX(90deg);
244   }
245 }
246 </style>
```