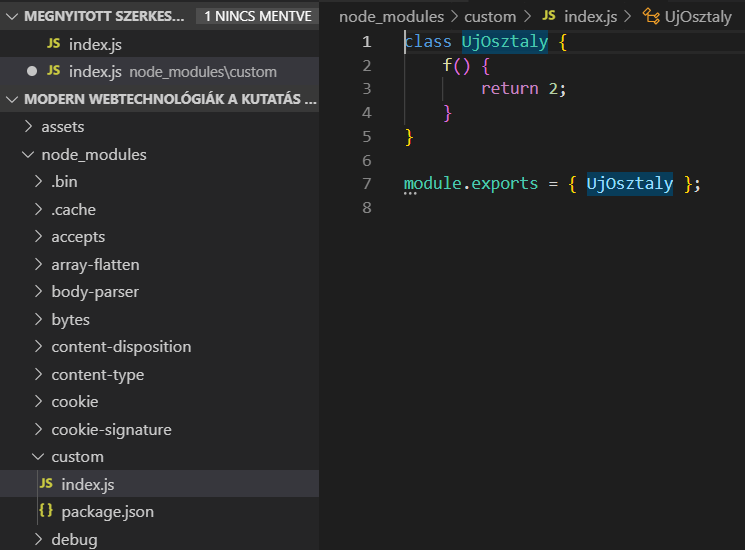
**Modern webtechnológiák a kutatás szolgálatában II.**

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.
* 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.



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

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

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 játékélményt, hogy változó sebességben jelennek meg csirkék a pályán, amiből ha túl sok lesz vesztettünk, illetve ha adott mennyiségű score-t gyűjtünk akkor gyorsul a játék (level up).

Megjegyezném, hogy először rosszul kezdtem el használi a vue-t, mivel teljesen js-esen fogtam fel (multimédia óráról használatos, documentGetElementByid() 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.

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

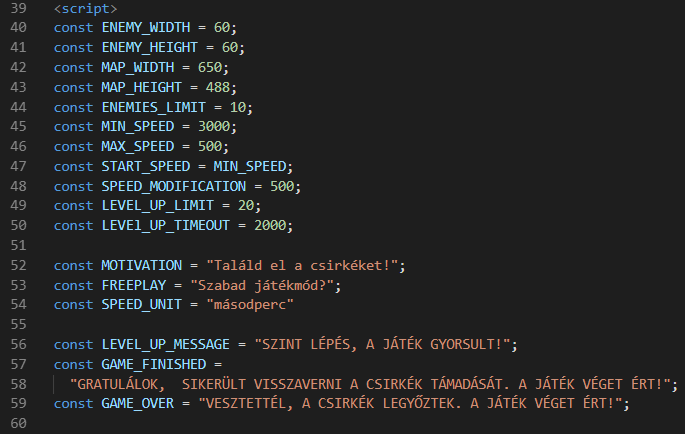
A **playground** bal felső sarkában egy piros számmal van jelezve, hogy hány csirke található a pályán akit le kellene lőni, a jobbfelső sarokban pedig zölddel, hogy már mennyit sikerült eltalálni az aktuális játékmenet alatt.

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 vesztünk, ez a szám konstansként van megadva a játék elején, mint ahogyan sok már is, így könnyen módosítható. Ez a szám most 10.



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



* 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.

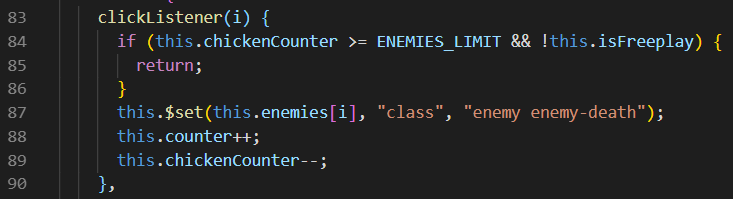
Ezt a data() metódusban tesszük meg:

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.

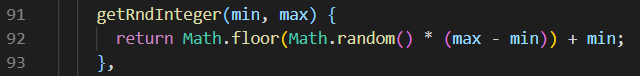
Most nézzük meg ezeket a metódusokat részletesebben:

clickListener(i):

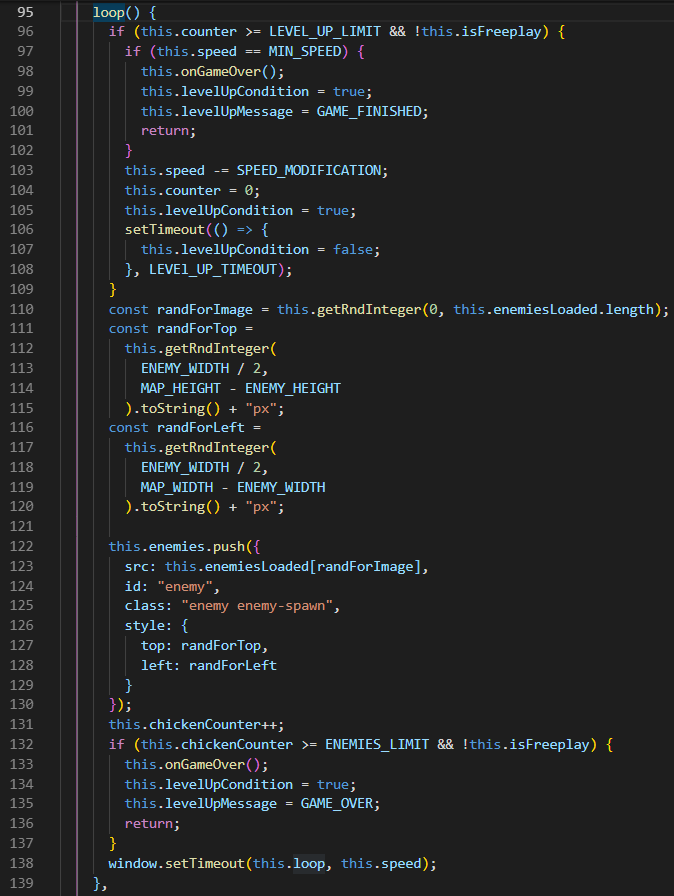


Vue jellegzetességéből adódóan egy img HTML taget rak ki egy ciklusban ami egy tömbből tölti be a csirkéket. Pontosan ezért ismert minden csirke fénykép tömb indexe, ezt használjuk itt i-ként, mint index és mint a for ciklus ciklusszámlálója. Ha a pályán lévő csirkék aktuális száma nagyobb vagy egyenlő 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 (chickenCoutner).

getRndInteger(min, max):



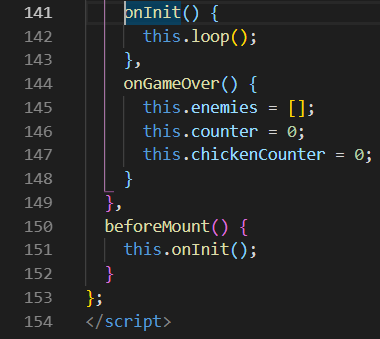
Erről nem hiszem, hogy sokat kell beszélni, alap JavaScript matematikus függvénytár metódusa, ami úgy van paraméterezve, hogy egyszerűen meglehessen neki mondani, hogy minimum és maximum mekkora szám lehet generálva.



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.

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.

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

A képen monitor, képernyő látható

Automatikusan generált leírásEzentúl még vannak az animációk, amik az utolsó képként jelennek meg a dokumentációban.

Előfordulhat, hogy későbbi push-ban történt némi változás, ami miatt picit módosult a kód, ami nem követi a kódról készült képeket.