



B8 – WEBOLDAL

Kovács Bálint

Tartalom

Bevezető:	3
Miért játék? Miért javascriptben?	4
GetData	4
REST API	5
FETCH API	6
Verziókezelés, publikálás	6
Munkamenet kezelés	7
AJAX	7
Gondolattérképek	8
UNIT TEST	9
Weboldal	10
CLIENT bejelentkezés.php	11
CLIENT- regisztracio.php	12
regisztracio, és bejelentkezés.php, és logout server oldalon	13
Snake	14
Ping Pong	17
STYLE.css	18
ball.js	18
SCRIPT.js	19
2024	21
Style.css	21
GRID.js	22
TILE.JS	23
SCRIPT.JS	24

Bevezető:

A webprogramozás mindig is a kedvencem volt, ezért úgy döntöttem ,hogy egy weboldalt készíték el, méghozzá olyat ami pár JavaScriptes játékot tartalmaz.

Ezekkel a játékokkal játszottam régen és szerettem őket, szóval úgy gondoltam jó ötlet lenne őket egy weboldalra rakni, és akár másoknak is hasznos lehet ha egy weboldalon megakarja találni ezeket a játékokat.

A Snake nevű játék lényege, hogy egy kígyó a karakterünk és a jelen esetben monster energia italt kell felszedni egy pályán, amint egyet felvettünk a kígyónk növekedik +1-gyel, ha magunknak, vagy a falnak megyünk akkor a játék véget ér. A cél, hogy minél nagyobb legyen a kígyó.

A Ping Pong nevű játékom lényege, hogy van egy platformunk a játék pálya egyik szélén, és egy botnak a másik szélén, és nem szabad ,hogy a labda megérintse a pálya szélét. Amelyik oldalon hozzáér a széléhez az ellenkező játékos kap +1 pontot.

A weboldalt HTML (HyperText Markup Language), CSS (Cascading Style Sheets), JS (JavaScript), és PHP programozási nyelvek segítségével készítem el.

A HTML egy leíró nyelv, melyet weboldalak készítéséhez fejlesztettek ki, és mára már internetes szabvánnyá vált.

A **CSS** (*Cascading Style Sheets*, magyarul: „lépcsőzetes stíluslapok”) a számítástechnikában egy stílusleíró nyelv, mely a HTML vagy XHTML típusú strukturált dokumentumok megjelenését írja le. Ezenkívül használható bármilyen XML alapú dokumentum stílusának leírására is, mint például az SVG, XUL stb.

A JavaScript programozási nyelv egy objektumorientált, prototípus alapú szkriptnyelv.

A PHP egy általános szerveroldali szkriptnyelv dinamikus weboldalak készítésére. az első szkriptnyelvek egyike, amely külsőfájl használata helyett HML oldalba ágyazható. A kódot a webszerver PHP feldolgozómodulja értelmezi, ezzel dinamikus weboldakat hozva létre.

Én a Visual Studio Code programot használtam, ezen belül hoztam létre minden játékot. A Visual Studio Code egy ingyenes, nyílt forráskódú kódszerkesztő, melyet a Microsoft fejleszt Windows, Linux és OS X operációs rendszerekhez. Támogatja a hibakeresőket, valamint beépített Git támogatással rendelkezik, továbbá képes az intelligens kódkiegészítésre az IntelliSense segítségével. A VSCode az Electron nevű keretrendszeren alapszik, amellyel asztali környezetben futtatható Node.js alkalmazások fejleszthetők. A Visual Studio Code a Visual Studio Online szerkesztőn alapszik (fejlesztési neve: "Monaco").

Egy képszerkesztő program egy olyan számítástechnikai szoftver, mely grafikusan megjelenített tartalom készítésére és módosítására alkalmas. A szerkesztett, készített képet bizonyos fájlformátumokba képes exportálni, melyek támogatottsága függ az adott szoftvertől

Miért játék? Miért javascriptben?

Úgy gondolom ,hogy játékos weboldal készítése sokkal közelebb áll hozzám mint például el webshop, már amikor megtudtam ,hogy záróvizsgát kell készíteni akkor tudtam ,hogy ezt szeretném csinálni, pár játékot szerettem volna készíteni amik azért viszonylag ismertek, és ezeket felrakni egy weboldalra, amit szintén én készítettem. Javascriptet amióta tanulom sokkal jobban megfogott mint bármi másmilyen nyelv, és otthon is könnyen tudom csinálni a munkám, hisz még egy nagyon gyenge laptopon is könnyeb elfut a Visual Studio Code, pár másik osztálytársam is Javascriptes záródolgozatot csinál, szóval még egymást is kitudtuk segíteni, a Javascript szerintem sokkal átláthatóbb mint a többi programozó nyelv, nagyon szimpatikus is számomra, a weboldal készítést is szeretem, szóval ez volt a tökéletes választás a számomra.

Elsőnek is a weboldalt hoztam létre, majd amikor az már nagyjából kész volt, elkezdtem a játékokat kódolni.

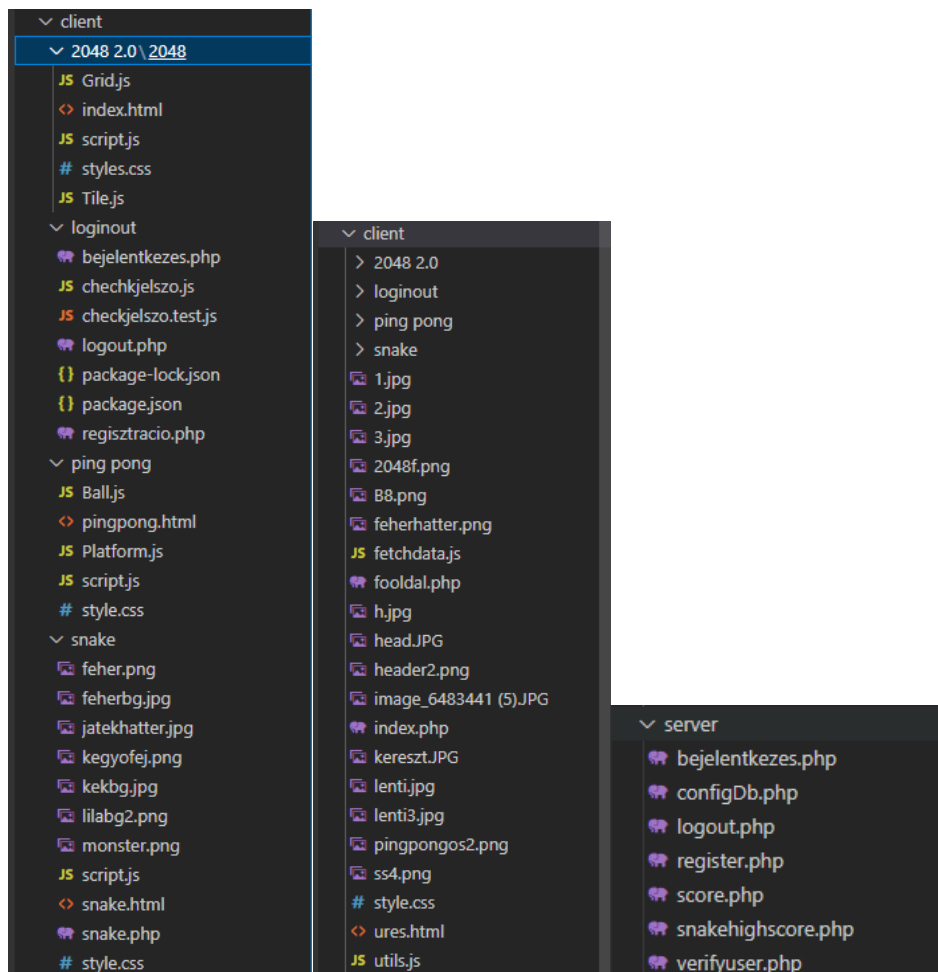
Viszonylag könnyebb játékokon gondolkoztam amiket kéne csinálni, a snake rögtön beugrott, a másik kettőhöz már kicsit elkellett gondolkodnom, hogy mi legyen és miként csináljam, végül egy ping pong és egy 2024 nevezetű játékra esett a választás.

GetData

```
const getdata=async (url,renderFc)=>{  
  const response=await fetch(url)  
  const data=await response.json()  
  renderFc(data)  
}
```

Ezzel tudunk adatokat lekérni phpből, json típussal. Asyncel jelezzük ,hogy ez egy hosszú művelete, majd await miatt megvárja a választ, és aztan visszaadja az adatot.

REST API



A REST API (Representational State Transfer API) egy olyan webes alkalmazásprogramozási felület, amely segítségével az alkalmazások kommunikálhatnak egymással az interneten keresztül.

Az API egy olyan interfész, amely lehetővé teszi, hogy különböző alkalmazások együttműködjenek egymással. Az API-k lehetnek egyszerűek, amelyek csak egyetlen funkciót kínálnak, vagy komplexek, amelyek többféle funkciót tartalmaznak.

A REST (Representational State Transfer) pedig egy architektúráis stílus, amely definiálja, hogy az alkalmazások hogyan kommunikáljanak egymással az interneten keresztül. Az alapelv az, hogy minden erőforrásnak egyedi azonosítója (URL) van, amelyen keresztül elérhető. Az erőforrások állapotát pedig a HTTP metódusok (GET, POST, PUT, DELETE stb.) segítségével lehet módosítani.

A REST API használata előnyös, mert lehetővé teszi, hogy az alkalmazások közötti kommunikáció egyszerű és hatékony legyen. Az API-k lehetővé teszik az adatok megosztását és a funkciók újrafelhasználását, ami nagyban javítja az alkalmazások integrációját és fejlesztését.

FETCH API

A Fetch API egy modern JavaScript API, amely lehetővé teszi a kliensoldali JavaScript számára, hogy HTTP kéréseket küldjön és fogadjon válaszokat a szerveroldali API-val. A Fetch API segítségével a böngészőből egyszerűen hozzáférhetünk az adatokhoz és az erőforrásokhoz a szerveren, így lehetővé teszi az aszinkron adatkérés küldését és fogadását.

A Fetch API alkalmazása előnyösebb lehet, mint például az XMLHttpRequest, mert sokkal egyszerűbb és modernabb módon lehet vele kommunikálni a szerverrel. A Fetch API lehetővé teszi a JavaScript kód számára, hogy írja át az oldalt aszinkron módon, anélkül, hogy az oldalt teljesen újra töltené.

A Fetch API használatával a kliensoldali JavaScript kód küldhet kéréseket a szerveroldali API-nak, amelyek JSON formátumban küldhetők és fogadhatók, ami könnyen kezelhető és kompatibilis a legtöbb modern programozási nyelvvel és adatbázissal.

A Fetch API segítségével a kliensoldali JavaScript kód egyszerűen kommunikálhat a szerveroldali API-val, és az aszinkron kommunikáció lehetővé teszi az oldalak dinamikusabbá és felhasználóbarátabbá tételét.

Verziókezelés, publikálás

A verziókezelés egy fontos eszköz a szoftvertervezésben, amely segít nyomon követni a kódbázis változásait, és biztosítja, hogy az előző verziókhoz vissza lehet térni, ha szükséges.

A verziókezelő rendszerek segítségével a fejlesztők könnyen nyomon követhetik a változásokat a forráskódban, illetve könnyen visszatérhetnek korábbi verziókhoz, ha szükséges. A legnépszerűbb verziókezelő rendszerek közé tartozik a Git, amit én is használtam, a Git könnyen elérhető mindenhol, az iskolából, és otthonról is.

A publikálás során a fejlesztők a kész szoftvert teszik elérhetővé az ügyfelek vagy felhasználók számára. A publikálás előtt általában tesztelik és ellenőrzik a szoftvert, hogy biztosak lehessenek benne, hogy az megfelelően működik. A publikálás lehet helyi (pl. saját szerveren) vagy távoli (pl. felhő alapú szolgáltatások) alapú. A publikálás lehet automatikus, amely az új kód változásait automatikusan publikálja, vagy manuális, amely az új kód változásait az üzemeltetőnek kell manuálisan publikálnia.

Munkamenet kezelés

A kliensek megkülönböztetését és a kliensenkénti adattárolást munkamenet-kezelésnek hívjuk. A HTTP protokoll állapotmentességének másik következménye, hogy hogyan tudjuk a klienseket megkülönböztetni egymástól? Hol tudjuk a kliensenkénti állapotot tárolni? Az oldalkiszolgálásban kliens és szerver vesz részt a HTTP protokoll segítségével. Mivel ezek közül a HTTP protokoll állapotmentes, ezért az állapot megtartását vagy kliens, vagy szerver oldalon oldhatjuk meg.

PHP-ban számos függvény és nyelvi elem segíti a munkamenet-kezelést. A PHP alapvetően elrejtja a munkamenet-azonosítóval kapcsolatos teendőket: automatikusan létrehozza őket, leküldi a kliensre, megfelelő módon váltogat a süti és az URL-es tárolás között, és egyszerű módon tárolja a munkamenetben tárolt adatokat. Ezek közül a fejlesztő számára ez utóbbi a legfontosabb. A munkamenet adatait a PHP a `$_SESSION` szuperglobális asszociatív tömbben tárolja. A szkript kezdetekor ebbe tölti be, a szkript végén pedig ennek tartalmát menti ki (alapértelmezetten fájlba).

AJAX

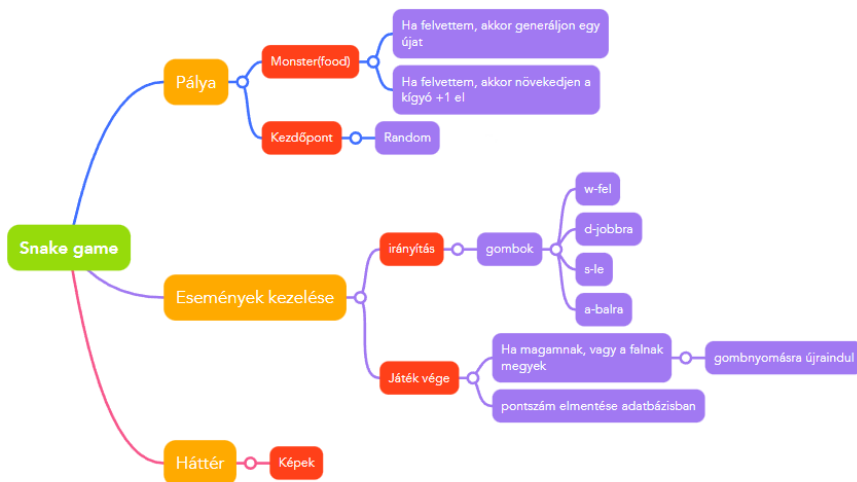
Az AJAX (Asynchronous JavaScript and XML) előnye, hogy lehetővé teszi, hogy egy weboldal kommunikáljon a szerverrel anélkül, hogy az oldal teljesen újratöltené magát. Ez azt jelenti, hogy amikor egy felhasználó valamilyen interakciót hajt végre az oldalon (pl. egy űrlap beküldése), az AJAX segítségével a weboldal csak a szükséges adatokat kéri le a szerverről, és azokat dinamikusan beilleszti az oldalra, anélkül, hogy az egész oldal újratöltődne.

Ez sok előnyt kínál az oldal látogatóinak, mivel az oldalak sokkal gyorsabban és zökkenőmentesebben működhetnek az AJAX használatával. Például, amikor egy felhasználó az oldalon keres valamit, az AJAX lehetővé teszi, hogy a keresési eredmények dinamikusan jelenjenek meg az oldalon, anélkül, hogy a teljes oldalt újratöltené.

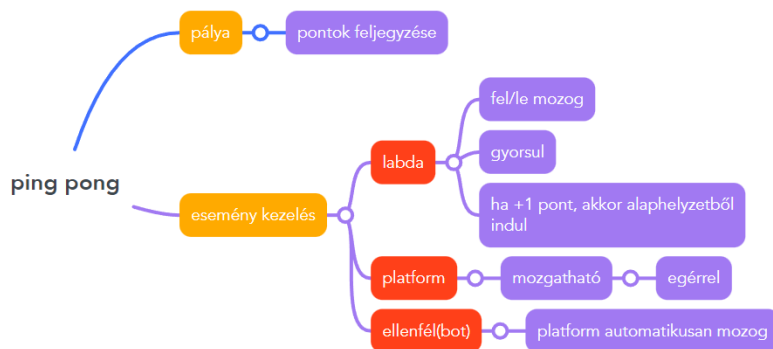
Az AJAX használatával az oldalak dinamikusabbak és interaktívabbak lehetnek, mivel az adatokat dinamikusan lehet betölteni és megjeleníteni az oldalon. Ezáltal az oldalak felhasználóbarátabbak lehetnek, és javíthatják a felhasználói élményt is.

Gondolattérképek

Gondolattérkép a Snake nevezetű játékhoz



Gondolattérkép a Ping Pong nevezetű játékhoz



Gondolattérkép a 2024 nevezetű játékhoz



UNIT TEST

```
zaroo > client > loginout > checkjelszo.test.js > ...
1  const { test, expect } = require('@jest/globals')
2  const checkhossz=require('./checkjelszo')
3
4  test("Jelszó hossz ellenőrzése",()=>{
5      let flag=checkhossz('123')
6      expect(flag).toBe(true)
7  })
8  test("Jelszó hossz ellenőrzése",()=>{
9      let flag=checkhossz('123456')
10     expect(flag).toBe(false)
11 })
```

```
PASS ./checkjelszo.test.js
  ✓ Jelszó hossz ellenőrzése (2 ms)
  ✓ Jelszó hossz ellenőrzése

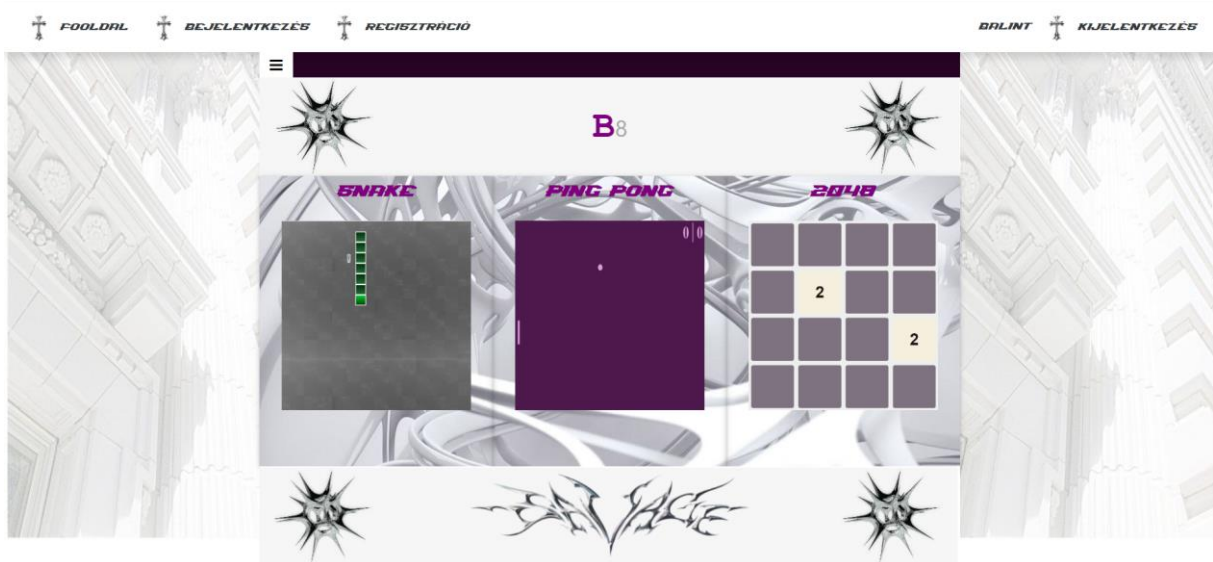
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.052 s
Ran all test suites.
PS C:\xampp\htdocs\zaro2\zaroo\client\loginout> █
```

Az adott kódrészlet egy JavaScript függvényt tartalmaz, amely "checkhossz" néven van elnevezve, és egy szöveget kap bemeneti paraméterként. A függvény ellenőrzi, hogy a bemeneti szöveg hossza kisebb-e, mint 6 karakter. Ha a hossz kevesebb, mint 6, akkor a függvény igaz értéket ad vissza. Ellenkező esetben hamis értéket ad vissza.

A kódban a Jest nevű keretrendszerrel tesztesetek vannak definiálva, amelyek a "checkhossz" függvényt tesztelik. Az első teszteset ellenőrzi, hogy a függvény igaz értéket ad-e vissza, ha a bemeneti szöveg hossza kevesebb, mint 6. A második teszteset ellenőrzi, hogy a függvény hamis értéket ad-e vissza, ha a bemeneti szöveg hossza 6 vagy annál több. A tesztesetek sikeresnek tekinthetők, ha a függvény kimenete megfelel az elvárt értéknek, ami az "expect" függvénnyel van definiálva.

Weboldal

A weboldalam lényege, hogy könnyen elbírjuk indítani az általam készített játékokat, és hogy átlátható legyen, a designelést az én tetszésemnek megfelelően készítettem el.



Az első három sorban három külső betűtípusfájl (font) hivatkozása található, amelyeket a weboldal az adott elemekhez (pl. címekhez, szövegekhez) használ.

```
<link href="https://fonts.cdnfonts.com/css/horror-type" rel="stylesheet">  
<link href="https://fonts.cdnfonts.com/css/vtks-rafia" rel="stylesheet">  
<link href="https://fonts.cdnfonts.com/css/spiky-016" rel="stylesheet">
```

Az "style" tagben található CSS kód definiálja a különböző stílusokat (pl. betűméret, betűtípus, háttérszín) a különböző elemekhez, például a címekhez, szövegekhez, gombokhoz és navigációs menükhöz.

A weboldalam tartalmaz egy navigációs menüt (nav tag) az oldal bal oldalán, amely néhány gombot tartalmaz (pl. Bejelentkezés, Regisztráció), amelyekre a felhasználó kattinthat. A header (fejléc) tartalmaz egy címet és egy képet a közepén, amely mögött egy háttérkép található.

Az oldal közepén három doboz (div tag) található, amelyekben három játék ikonja található, amelyekre a felhasználó kattinthat a játék megkezdéséhez. A játékok közé tartoznak a Snake, a Ping Pong és a 2048. A div tag alatt egy kép található, amely a lenti részét teszi ki az oldalnak.

Az oldal elején és végén található div tag-ekben található képek, amelyek dekoratív célokat szolgálnak, és nem közvetlenül kapcsolódnak az oldal tartalmához.

CLIENT bejelentkezés.php



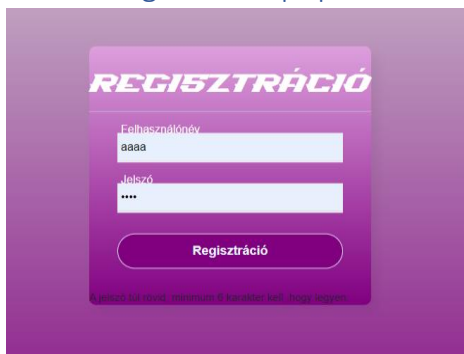
```
<div class="center">
  <h1>Belépés</h1>
  <form method="post">
    <div class="txt_field">
      <input type="text" name="username" required>
      <span></span>
      <label>Felhasználónév</label>
    </div>
    <div class="txt_field">
      <input type="password" name="password" required>
      <span></span>
      <label>Jelszó</label>
    </div>
    <input type="button" onclick="loginuser()" value="Belépés">
    <div class="signup_link">
      Még nem regisztráltál? <a href="index.php?prog=../loginout/regisztracio.php">Regisztráció</a>
    </div>
  </form>
</div>

<script>
function loginuser(){
  const myFormData = new FormData(document.querySelector('form'));
  let configObj={
    method: 'POST',
    body: myFormData
  }
  postdata('../server/bejelentkezés.php',configObj,render)
}

function render(data){
  console.log(data)
  if(data.msg=='ok')
    location.href='../index.php'
}
</script>
```

Ez a kód egy felhasználói felületet (UI) jelenít meg, amelyen keresztül a felhasználók be tudnak jelentkezni az alkalmazásba. A felhasználónév és a jelszó megadása után a loginuser() függvény meghívja az ../server/bejelentkezés.php elérési utat POST kéréssel a megadott adatokkal. A render() függvény válaszként kapott adatokat kezeli, amelyeket a szerver küld vissza JSON formátumban. Ha a válasz "ok", akkor az oldal átirányítja a felhasználót a ../index.php oldalra. Ha valami hiba történt, akkor a hibaüzenetet a konzolon jeleníti meg. A HTML és JavaScript kódrészek interakciója révén a felhasználók be tudnak jelentkezni az alkalmazásba.

CLIENT- regisztracio.php



The screenshot shows a web registration form with a purple gradient background. The form is centered and contains the following elements:

- A title "REGISZTRÁCIÓ" in a stylized, italicized font.
- A label "Felhasználónév" above a text input field containing the text "aaaa".
- A label "Jelszó" above a password input field containing four asterisks "****".
- A rounded button labeled "Regisztráció".
- A small message at the bottom: "A jelszó legalább 6 karakterből kell, hogy álljon."

```
<div class="center">
  <h1>Regisztráció</h1>
  <form method="post">
    <div class="txt_field">
      <input type="username" id="username" onblur="vrusername(this)" name="username" placeholder="Felhasználó név" required autofocus>
      <span></span>
      <label>Felhasználónév</label>
    </div>
    <div class="txt_field">
      <input onblur="checkhossz(this)" type="password" id="password" name="password" placeholder="Jelszó" required>
      <span></span>
      <label>Jelszó</label>
    </div>
    <input type="button" class="register" onclick="newUser()" value="Regisztráció"></button>
    <div class="signup_link">
    </div>
  </form>
  <div class="msg"></div>
</div>
```

Ez a kód egy űrlapot jelenít meg egy weboldalon a felhasználó regisztrációjához. Az űrlap két mezőt tartalmaz, a felhasználónév és a jelszó megadására szolgál. Az űrlap elküldéséhez szükséges "Regisztráció" gombra kattintani, amely egy JavaScript függvényt hív meg. Az űrlap ellenőrzését szintén JavaScript végzi, és hibaüzenetet jelenít meg, ha a jelszó rövidebb, mint hat karakter. A regisztráció sikeres befejezése után a felhasználó bejelentkezhetsz a "Jelentkezz be!" gombra kattintva, ami visszairányítja a felhasználót a bejelentkezési oldalra.

regisztracio, és bejelentkezés.php, és logout server oldalon.

```
<?php
session_start();
require_once 'configDb.php';
extract($_POST);
$pw=password_hash($password,PASSWORD_DEFAULT);
$sql="INSERT INTO users (username,password) VALUES ('{$username}','{$pw}')";
try{
    $stmt=$db->exec($sql);
    echo json_encode(["msg"=>"Sikeres regisztráció! azonosító!", "id"=>$db->lastInsertId()]);
}catch(exception $e){
    echo json_encode(["msg"=>"Sikertelen regisztráció! {$e}"]);
}
?>
```

Ez egy PHP szkript, amely egy új felhasználót regisztrál az adatbázisban. Ha a kísérlet sikeres volt, akkor egy JSON objektumot ad vissza a szkript, amely a sikeres regisztráció üzenetet és az új felhasználó azonosítóját tartalmazza. Ha a kísérlet nem sikerült, akkor egy hibaüzenetet ad vissza a szkript, amely tartalmazza az okát, hogy miért nem sikerült a regisztráció.

```
<?php
session_start();
require_once 'configDb.php';
extract($_POST);

$sql="SELECT PASSWORD pwcrypted from users where username='{$username}'";

try{
    $stmt=$db->query($sql);
    if($stmt->rowCount()==1){
        $row=$stmt->fetch();
        extract($row);
        $isValid=password_verify($password,$pwcrypted);
        if($isValid){
            $_SESSION['username']=$username;
            echo json_encode(["msg"=>"ok"]);
        }else{
            echo json_encode(["msg"=>"Hibás felhasználónév/jelszó"]);
        }else{
            echo json_encode(["msg"=>"Hibás felhasználónév"]);
        }catch(exception $e){
            echo json_encode(["msg"=>"Sikertelen regisztráció! {$e}"]);
        }
    }
?>
```

Ez a PHP kód a felhasználói bejelentkezés funkcióval kapcsolatos.

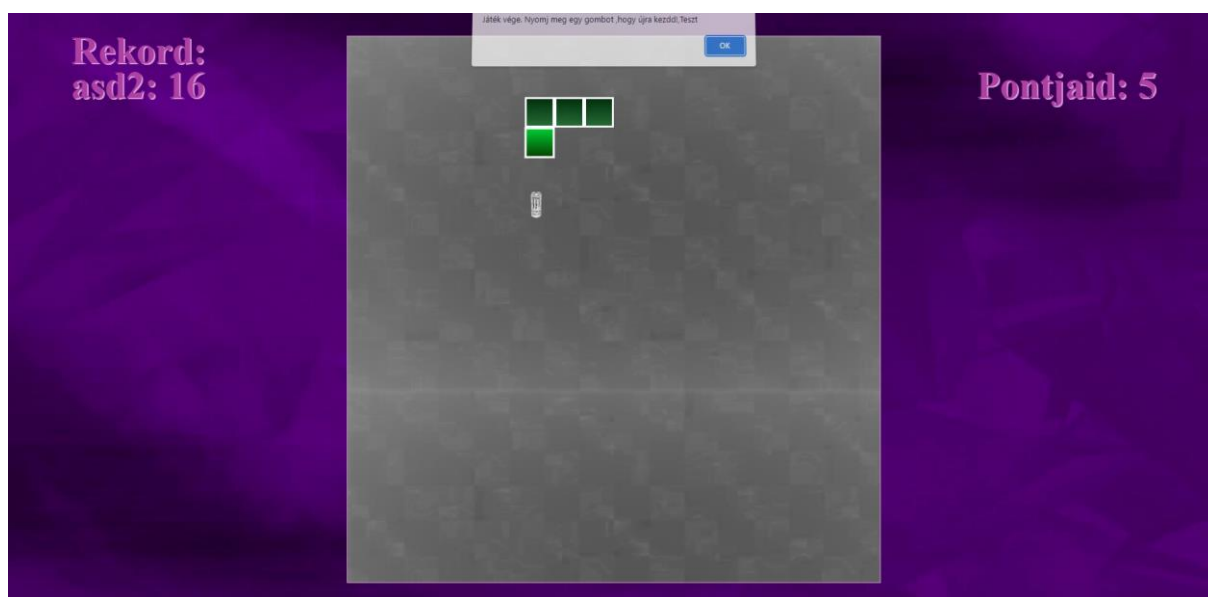
```
<?php
session_start();
unset($_SESSION['username']);
session_destroy();
echo true;
?>
```

Ez a PHP kód a felhasználó kijelentkeztetésére szolgál.

Snake

A snake játékom lényege, hogy egy kígyót kell irányítani a nyilakkal, és minél több monstert kell felvenni a pályáról, ha felvettünk egyet, akkor a kígyónk nagysága növekedni fog +1 el, és ezzel párhuzamosan a pontjaink is növekedik. A játéknak akkor lesz vége, ha nekimegyünk a falnak, vagy pedig saját magunknak megyünk neki, amit egyre nehezebb lesz elkerülni, ha a kígyónk már nagy.

Így néz ki a játékom, van a pálya része, amin a kígyó tud mozogni és az étel van, ami jelen esetben egy monster energia ital, a rekord tartó nevét, és pontszámát a bal oldalon jelzi, a jelenlegi pontjaim pedig a jobb oldalt, amikor a játéknak vége, egy üzenetet kapunk, ami elmondja, hogy ha új játékot szeretnék játszani akkor, csak nyomjunk egy billentyűt le.



Minden egyes játék után a felhasználó neve, és a pontjai elmentődnek az adatbázisban. Az adatbázis tábla a következő attribútumokat használja:

id: a tábla elsődleges kulcsa, azonosítja az adott sort.

username: a weboldalon használt felhasználó név

score: a játékban megszerzett pontokat menti el, minden egyes játék után

createdat: megadja a pontos időt, hogy mikor volt játszva.

A tábla neve highscore, a célja, hogy kiirassuk a legjobb eredményt.

Főbb elemei a játéknak:

```
if(snakeArray[0].y == food.y && snakeArray[0].x == food.x){
    snakeArray.unshift({x: snakeArray[0].x + irany.x, y: snakeArray[0].y + irany.y})
    let a =2;
    let b=16;
    food = {x: Math.round(a + (b-a) * Math.random()),y: Math.round(a + (b-a) * Math.random())};
    score +=1;
    scorebox.innerHTML = "Pontjaid: " + score;
}
```

Ha a snakeArray tömb első eleme (snakeArray[0]) a food változóval azonos koordinátákat (food.x és food.y) mutat, akkor a kígyó feje ütközött az étellel. Ekkor a kód hozzáad egy új elemet a snakeArray tömb elejéhez a unshift() metódus segítségével. Az új elem x koordinátája a fej x koordinátájához hozzáadva a mozgás irányvektorának x komponensét (irany.x), míg az y koordinátája a fej y koordinátájához hozzáadva a mozgás irányvektorának y komponensét (irany.y). Ez az új elem egy lépéssel elmozdítja a kígyó fejét az aktuális irányba. Ezután a kód új véletlenszerű koordinátákat generál az ételnek, frissíti a pontszámot (score) és egy HTML elemet (scorebox), majd folytatja a program futását. A kód közvetlenül frissíti a táblát.

```
window.addEventListener('keydown', (e)=> {
    irany = {x: 0, y: 1} //indul
    switch(e.key){
        case "ArrowUp":
            // console.log('ArrowUp')
            irany.x=0;
            irany.y=-1;
            break;
        case "ArrowDown":
            //console.log('ArrowDown')
            irany.x=0;
            irany.y=1;
            break;
        case "ArrowLeft":
            //console.log('ArrowLeft')
            irany.x=-1;
            irany.y=0;
            break;
        case "ArrowRight":
            //console.log("ArrowRight")
            irany.x=1;
            irany.y=0;
            break;
        default:
            irany ={x:0, y:0}
            console.log('Más billentyű')
            return
    }
})
```

Irányítás, csak is a billentyűkkel mozog, ha mást nyomunk semmi nem fog történni, ezt a keydown eventlistener eseménnyel oldottam meg, ez a módszer nagyon könnyű és átlátható, beírtam a megfelelő nyilat és egy irányt, hogy merre mozogjon.

```
function jatekvege(snake){

    for(let i = 1; i<snakeArray.length;i++){
        if(snake[i].x== snake[0].x && snake[i].y == snake[0].y){
            return true;
        }
    }
}
```

Ez a kódrészlet egy JavaScript függvény, amely a játék végét ellenőrzi. A függvény paraméterként kapja a kígyó helyzetét reprezentáló tömböt (snake), majd visszatérési értéként true-t ad vissza, ha a kígyó elpusztult, vagy false-t, ha még nem.

A függvény a következőket teszi:

- Először kikommentezett egy `return false;` sort, amely mindig false értéket ad vissza. Ha a függvénynek nincs más feladata, akkor ez a sor kikommentezése nem változtat semmit.
- A függvény végigmegy a snakeArray tömb elemein (azaz a for ciklus i változója 1-től kezdve az utolsó elemig megy), és ellenőrzi, hogy a jelenlegi elem (snake[i]) azonos koordinátákkal rendelkezik-e, mint a kígyó feje (snake[0]). Ha igen, akkor a kígyó feje beleütközött az egyik saját testrészébe, így a függvény true értéket ad vissza.
- A függvény azt is ellenőrzi, hogy a kígyó feje (snake[0]) kilépett-e a játéktéren belülre (azaz a koordinátái nagyobbak vagy kisebbek-e, mint a játéktér méretei). Ha igen, akkor a kígyó elpusztult, és a függvény true értéket ad vissza.
- Ha a függvény eddig nem adott vissza értéket, akkor a kígyó még mindig él, így a függvény false értéket ad vissza.

```
background.innerHTML = "";
snakeArray.forEach((e, index)=> {
    snakeElement = document.createElement('div');
    snakeElement.style.gridRowStart = e.y;
    snakeElement.style.gridColumnStart = e.x;
    snakeElement.classList.add('snake')
    background.appendChild(snakeElement);

    if(index ==0){
        snakeElement.classList.add('head');
    }else{
        snakeElement.classList.add('snake');
    }
});
```

Ez a kódrészlet a kígyó mozgását rajzolja ki a képernyőn. Először törli a háttéren lévő összes elemet, majd a kígyó minden egyes eleméhez létrehoz egy DIV elemet és beállítja annak helyzetét a CSS grid-rendszer használatával (a gridRowStart és a gridColumnStart stílusokat használja ehhez). A DIV elemet a snake osztályba helyezi, majd hozzáadja a háttérhez (background).

Ha az index értéke 0, a head osztályt is hozzáadja az elemhez, hogy megjelenítse a kígyó fejét. Ha az index nem 0, akkor csak a snake osztályt adja hozzá.


```
for(let obj of data){  
    Str+=`  
        <tr>  
            <td>${obj.username}</td>  
            <td>${obj.score}</td>  
        </tr>  
    `;  
}
```

ez a kód egy adatbázisból lekérdezett adatokat jelenít meg HTML táblázat formájában. A táblázat minden sorában a felhasználónevet és a pontszámát jeleníti meg. A data tömb elemein egy cikluson keresztül megy á, és minden egyes elemhez hozzáad egy új táblázatsort, amelynek két cellája van: az egyikben a felhasználónév, a másikban a pontszám található.

Ping Pong

A ping pong játék lényege, hogy a miénk a bal oldali platform, és a labdát kikell védeni, nem szabad hogy hozzáérjen a mi oldalunkhoz, ha a labda valamelyik oldalhoz hozzáért, akkor az ellenkező oldal kap +1 pontot, minél tovább védjük ki a labdát annál nehezebb lesz a játék, mivel folyamatosan gyorsul a labda, a bot sem tud mindent kivédeni.

Így néz ki a játék, van 2 platform, középen felül egy számláló, és a labda.



[STYLE.css](#)

Ping Pong játék vizuális megjelenítéséhez szükséges stílusszabályokat tartalmazza. A főbb részei a következők:

A box-sizing tulajdonságot minden elemre beállítja, így a számítások a kereteket is figyelembe veszik.

A :root szabályhalmazban változókat definiál a játék színeihez és háttérszínéhez. A színek HSL színskála értékeivel vannak megadva.

A body elemhez alapértelmezett stílusokat ad, például a margót eltávolítja, a háttérszínt beállítja és az overflow tulajdonságot hidden-re állítja, hogy az elemen kívüli tartalmak ne jelenjenek meg a játékban.

A paddle osztály definiálja a játékosokhoz tartozó pingpong ütők stílusát. A --position változóval állítható be az ütő pozíciója, és az absolute elhelyezkedési tulajdonsággal van beállítva.

A ball osztály definiálja a labda stílusát. A --x és --y változók állítják be a labda pozícióját, és a border-radius tulajdonsággal kerekített határozott alakot kap.

A score osztály definiálja a pontszám megjelenítését. Az align-items tulajdonsággal középre igazítja az elemeket, és az flex-grow és flex-basis tulajdonságokkal beállítja az elemek méretét és elrendezését. A border-right tulajdonsággal a pontszám oszlopait elválasztó sávot hozza létre.

[ball.js](#)

Ez a függvény egy "Ball" osztályt definiál, amely a játék labdájának viselkedését írja le. A "constructor" függvény beállítja a "ballElem" tulajdonságot és visszaállítja a labda pozícióját az eredeti helyzetébe. A "get" és "set" függvények az "x" és "y" tulajdonságokat írják le, amelyek visszaadják és beállítják a labda aktuális pozícióját. Az "platform" függvény a labdát tartalmazó DOM elem méretét és pozícióját adja vissza. A "reset" függvény visszaállítja a labdát az eredeti pozícióba, majd véletlenszerű irányba indítja el a labdát a "direction" objektum segítségével, majd beállítja a mozgás sebességét is. Az "update" függvény a labda pozícióját frissíti az idő függvényében, majd ellenőrzi, hogy a labda érintkezik-e a falakkal vagy a játékos platformjaival, és megváltoztatja a labda mozgási irányát, ha szükséges. A "randomszam" függvény véletlenszerű számokat generál, a "hossaer" függvény pedig ellenőrzi, hogy két DOM elem érintkezik-e egymással.

```
update(ido, paddlerects){
  this.x += this.direction.x * this.mozgas * ido
  this.y += this.direction.y * this.mozgas * ido
  this.mozgas += mozgasgyorsulas * ido
  const platform = this.platform()

  if(platform.bottom >= window.innerHeight || platform.top <= 0){
    this.direction.y *= -1
  }
  if(paddlerects.some(r => hossaer(r,platform))){
    this.direction.x *= -1
  }
}
}
function randomszam(min, max){
  return Math.random() * (max - min) + min
}
function hossaer(platform1, platform2){
  return platform1.left <= platform2.right && platform1.right >= platform2.left && platform1.top <= platform2.bottom && platform1.bottom >= platform2.top
}
}
```

```
const kezdogyors = .025
const mozgasgyorsulas = 0.00001
```

```
get x() {
  return parseFloat(getComputedStyle(this.ballElem).getPropertyValue("--x"))
}
set x(value) {
  this.ballElem.style.setProperty("--x",value)
}
get y() {
  return parseFloat(getComputedStyle(this.ballElem).getPropertyValue("--y"))
}
set y(value) {
  this.ballElem.style.setProperty("--y",value)
}
platform() {
  return this.ballElem.getBoundingClientRect()
}
```

SCRIPT.js

A kód a Ball és a Paddle osztályokat használja a labda és a platformok viselkedésének leírására. A pong játék célja az, hogy a labdát visszapattintva elkerüld a veszteséget, és pontokat szerezz. Ha a labda a játékos szélén landol, az ellenfél pontot szerez, és a játék újakezdődik. A játékot az egér mozgásával lehet játszani, és a pontszámot a játékos két oldalán található számok jelzik.

```
function ujra(){
  const platform = ball.platform()
  if (platform.right >= window.innerWidth){
    playerscore.textContent = parseInt(playerscore.textContent) + 1
  } else {
    computerscore.textContent = parseInt(computerscore.textContent) + 1
  }
  ball.reset()
  computerPaddle.reset()
}
```

```
let lastTime
function update(time) {
  if (lastTime !== null) {
    const ido = time - lastTime
    ball.update(ido, [playerPaddle.platform(), computerPaddle.platform()])
    computerPaddle.update(ido, ball.y)

    if(vesztes()) ujra()
  }

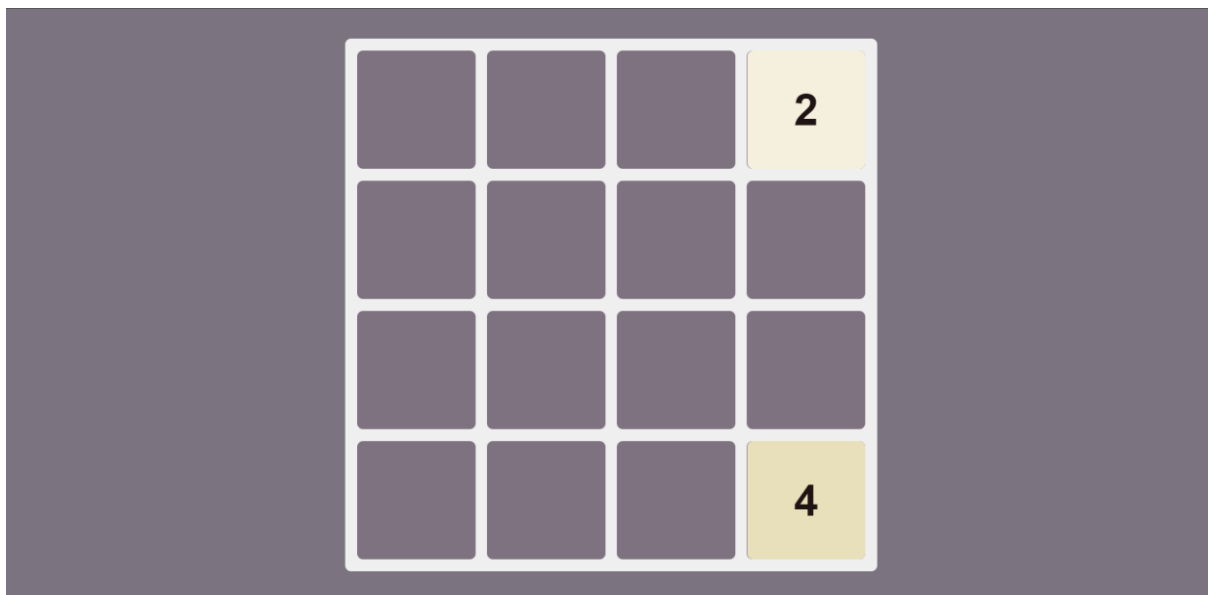
  lastTime = time
  window.requestAnimationFrame(update)
}
```

```
document.addEventListener("mousemove", e => {
  playerPaddle.position = (e.y / window.innerHeight) * 100
})
```

Egérrel irányítjuk a platformot, amit egy mousemove eventlistener eseménnyel hoztam létre, és aztán ezt méreteztem.

2024

A 2024 egy táblás számjáték, amelyben a játékosok egy négyzetrácsos játéktáblán mozgatnak számokkal teli csempeket. Az alapjátékban két csempe található a táblán, az értékük 2 vagy 4 lehet. A játékosok felváltva mozgatják a csempeket a négy irányba (fel, le, jobbra, balra), és azonos értékű csempek összeolvadnak, hogy egy új csempe jöjjön létre, amelynek értéke az összeolvasztott csempek összege. Az a cél, hogy az egyes csempek értéke mindél több legyen. Azonban a játék véget ér, ha megtelik a tábla, és a játékosok nem tudnak további összeolvadásokat végrehajtani. A játék nehézségi szintje változhat azáltal, hogy a táblán lévő kezdeti csempek száma, értékei vagy a játékosok mozgásainak száma változtatható.



Style.css

```
*, *::before, *::after {  
  box-sizing: border-box;  
  font-family: Arial;  
}
```

Ez egy CSS stíluslap, amely egy játéktáblát formázza meg. A stíluslap egyetemes CSS tulajdonságokat határoz meg (*, *::before, *::after), mint a dobozmodell és a betűtípus, majd a testet (body) formázza meg, amely egy középre igazított játéktáblát tartalmaz. A játéktábla (id="game-board") egy rácsszabályt tartalmaz, amely a cellákat és a cellaméretet határozza meg. A cellák és a csempek (tiles) egyedi formázást kapnak, a csempek animációval jelennek meg és átmenetet használnak az egyszerűbb áttűnésekhez. A színek hsl kóddal vannak megadva, amelyek lehetővé teszik a színárnyalatok finom beállítását a teljesen különböző színárnyalatok érdekében.

GRID.js

A Grid (rács) osztály reprezentálja a játéklemezőt, amely cellákból áll, és kezeli azokat. A Cell (cella) osztály pedig reprezentálja az egyes cellákat a játéklemezőn, és a Tile (csempe) osztály pedig reprezentálja a számokat tartalmazó csempéket, amelyeket a játékos helyezhet el a cellákban. A játék a billentyűzet segítségével irányítható, és a cél az, hogy a játékos a lehető legnagyobb számot érje el a cellákban lévő csempék összeadásával. A kód megvalósítja a csempék mozgását és összevonását is, és véget ér, ha a játékos nem tud tovább lépni.

```
constructor(gridElement) {  
  gridElement.style.setProperty("--grid-size", GRID_SIZE)  
  gridElement.style.setProperty("--cell-size", `${CELL_SIZE}vmin`)  
  gridElement.style.setProperty("--cell-gap", `${CELL_GAP}vmin`)  
  this.#cells = createCellElements(gridElement).map((cellElement, index) => {  
    return new Cell(  
      cellElement,  
      index % GRID_SIZE,  
      Math.floor(index / GRID_SIZE)  
    )  
  })  
}
```

Ez a Grid osztály konstruktor, amely létrehozza a játéktábla celláit (a Cell objektumokat), és beállítja a megfelelő CSS változókat a játéktáblához tartozó DOM elemen. A konstruktor megkapja a játéktáblához tartozó DOM elemet (gridElement), és beállítja a "--grid-size", "--cell-size" és "--cell-gap" CSS változók értékeit, amelyek meghatározzák a játéktábla méretét és a cellák közötti hézagot.

A konstruktor továbbá meghívja a createCellElements függvényt, amely létrehozza a DOM cellákat, és a Cell osztály konstruktorát hívja meg az új Cell objektumok létrehozásához. A Cell objektumokat egy tömbbe gyűjti (this.#cells), és beállítja a Grid osztály privát #cells mezőjébe.

TILE.JS

```
constructor(tileContainer, value = Math.random() > 0.5 ? 2 : 4) {
  this.#tileElement = document.createElement("div")
  this.#tileElement.classList.add("tile")
  tileContainer.append(this.#tileElement)
  this.value = value
}

get value() {
  return this.#value
}

set value(v) {
  this.#value = v
  this.#tileElement.textContent = v
  const power = Math.log2(v)
  const backgroundLightness = 100 - power * 9
  this.#tileElement.style.setProperty(
    "--background-lightness",
    `${backgroundLightness}%`
  )
  this.#tileElement.style.setProperty(
    "--text-lightness",
    `${backgroundLightness <= 50 ? 90 : 10}%`
  )
}

set x(value) {
  this.#x = value
  this.#tileElement.style.setProperty("--x", value)
}

set y(value) {
  this.#y = value
  this.#tileElement.style.setProperty("--y", value)
}

remove() {
  this.#tileElement.remove()
}
```

Ez egy JavaScript osztálydefiníció egy Tile objektumhoz. A Tile egyetlen mezőt képvisel a játéktáblán olyan játékokban, mint a 2048. Az osztály privát mezőket használ, amelyeket a "#" szimbólum jelöl.

A konstruktor egy tileContainer elemet és egy opcionális értéket vesz át a mezőhöz. Ha nem adunk meg értéket, akkor véletlenszerűen generál egy 2 vagy 4 értéket. Létrehoz egy új div elemet, amely a mezőt reprezentálja, és hozzáadja azt a tileContainer elemhez.

A Tile osztálynak több getter és setter metódusa van, amelyekkel lekérhetjük vagy beállíthatjuk a tulajdonságok értékét: #tileElement, #x, #y és

#value.

Az érték beállító metódus beállítja a mező értékét, és frissíti a mező elem szöveges tartalmát és háttérszínét az érték alapján.

Az x és y beállító metódusok beállítják a mező pozícióját a játéktáblán a tile elem "--x" és "--y" egyéni tulajdonságainak frissítésével.

A remove metódus eltávolítja a mező elemet a DOM-ból.

```
waitForTransition(animation = false) {
  return new Promise(resolve => {
    this.#tileElement.addEventListener(
      animation ? "animationend" : "transitionend",
      resolve,
      {
        once: true,
      }
    )
  })
}
```

A waitForTransition metódus egy ígéretet ad vissza, amely akkor teljesül, amikor a mező elem animációja vagy átmenete véget ér. Egy opcionális boolean argumentumot, "animation"-t is átvehet, amely alapértelmezetten hamis. Ha igazra állítjuk, akkor az animáció vége előtt várakozik, és nem az átmenet vége előtt.

SCRIPT.JS

```
import Grid from "../Grid.js"
import Tile from "../Tile.js"

const gameBoard = document.getElementById("game-board")

const grid = new Grid(gameBoard)
grid.randomEmptyCell().tile = new Tile(gameBoard)
grid.randomEmptyCell().tile = new Tile(gameBoard)
setupInput()

function setupInput() {
  window.addEventListener("keydown", handleInput, { once: true })
}
```

Ez a JavaScript kód modulokat importál Grid és Tile nevű osztályokból, majd példányosítja őket és hozzáadja a játéktáblához.

A "gameBoard" konstans az "id" attribútummal rendelkező "game-board" HTML elemet jelöli meg, amely a játéktábla tartalmazó elem.

A "Grid" és "Tile" modulok a játéktábla és a csempék (tiles) megjelenítéséért felelnek a játékban. A "Grid" objektum példányosítása először két csempét helyez el a játéktáblán véletlenszerűen kiválasztott üres cellákba a "randomEmptyCell()" metódus használatával, majd hozzáadja őket a táblához a "new Tile(gameBoard)" kóddal.

A "setupInput()" függvény a "keydown" eseményre vár és egyszer fut le. Amikor a felhasználó lenyomja a billentyűt, akkor meghívódik az "handleInput()" függvény, amely az irányításban fog segíteni.

```
grid.cells.forEach(cell => cell.mergeTiles())

const newTile = new Tile(gameBoard)
grid.randomEmptyCell().tile = newTile

if (!canMoveUp() && !canMoveDown() && !canMoveLeft() && !canMoveRight()) {
  newTile.waitForTransition(true).then(() => {
    alert("Vesztettél!")
  })
  return
}

setupInput()
}
```

Ez a rész a játék léptetéséért felelős. Először is az összes cellán végigmegyünk a forEach metódus segítségével, majd meghívjuk rájuk a mergeTiles metódust. Ez a metódus összefűzi az egyező értékű csempéket egy adott cellában. Ezután a játéktáblához hozzáadunk egy új csempét a newTile változóban lévő új Tile objektum segítségével, amit egy véletlenszerű üres cellába helyezünk el a randomEmptyCell metódus segítségével.

Ezután megvizsgáljuk, hogy az összes irányban van-e további lépés. Ha nincs, akkor az újonnan elhelyezett csempe animációjának végét várjuk meg a `waitForTransition` metódus segítségével. Amikor az animáció véget ér, megjelenítünk egy felugró ablakot a `alert` segítségével, amely jelzi, hogy vesztettél. Ha még lehet tovább lépni, akkor a `setupInput` függvény segítségével beállítjuk az eseménykezelőt a következő lépésre.

```
function slideTiles(cells) {
  return Promise.all(
    cells.flatMap(group => {
      const promises = []
      for (let i = 1; i < group.length; i++) {
        const cell = group[i]
        if (cell.tile == null) continue
        let lastValidCell
        for (let j = i - 1; j >= 0; j--) {
          const moveToCell = group[j]
          if (!moveToCell.canAccept(cell.tile)) break
          lastValidCell = moveToCell
        }

        if (lastValidCell != null) {
          promises.push(cell.tile.waitForTransition())
          if (lastValidCell.tile != null) {
            lastValidCell.mergeTile = cell.tile
          } else {
            lastValidCell.tile = cell.tile
          }
          cell.tile = null
        }
      }
      return promises
    })
  )
}
```

Ez egy JavaScript függvény definíció a játéktábla mezőinek mozgatásához. A függvény `Promise` objektumot ad vissza, és összes mezőt mozgat, majd várja, hogy az animációk és a tranzakciók befejeződjenek. A mozgatás során a mezők összevonódnak, ha megfelelőek az összevonás feltételei. A függvény belső működése során iterál az összes mezőn és a csoportokon, hogy meghatározza, mely mezők mozgathatóak a megadott irányba, majd végrehajtja azokat az animációkat és tranzakciókat, amelyek az adott mozgást eredményezik. Ha egy mező üres, akkor az animáció és a tranzakció nélkül ugrik át a következő mezőre. Ha a mező értéke megegyezik, akkor az összevonás megtörténik, és az eredményül kapott mező értéke az összevont mezők összegével lesz egyenlő. Ha a mezők értéke nem egyezik meg, akkor a mező mozgatható az utolsó, nem üres mezőig, majd a végén a mező értékét beállítják az utolsó nem üres mező értékére, és az üres mező értékét nullra állítják.

```
function canMoveUp() {
  return canMove(grid.cellsByColumn)
}

function canMoveDown() {
  return canMove(grid.cellsByColumn.map(column => [...column].reverse()))
}

function canMoveLeft() {
  return canMove(grid.cellsByRow)
}

function canMoveRight() {
  return canMove(grid.cellsByRow.map(row => [...row].reverse()))
}

function canMove(cells) {
  return cells.some(group => {
    return group.some((cell, index) => {
      if (index === 0) return false
      if (cell.tile == null) return false
      const moveToCell = group[index - 1]
      return moveToCell.canAccept(cell.tile)
    })
  })
}
```

Ez négy segédfüggvény, amelyek eldöntik, hogy lehet-e mozogni a táblán a felfele, lefele, balra vagy jobbra irányba. Ezek a függvények a `Grid` osztály `cellsByRow` és `cellsByColumn` tömbjeire támaszkodnak, amelyek a táblát sorokra vagy oszlopokra bontják fel. Ezeket a tömböket a `canMove` függvény használja, amely eldönti, hogy a csoportban található bármely cellában található csempe mozgatható-e egy olyan cellába, amely a csoporton belül van, de előtte üres volt, és amely elfogadja a csempét.

Felhasznált források:

1.

<https://hu.wikipedia.org/wiki/JavaScript>

2.

https://www.youtube.com/watch?v=fC3j2U_UZrQ&ab_channel=Mr.WebDesigner

3.

<https://stackoverflow.com/questions/>

4.

<https://hu.wikipedia.org/wiki/HTML>

5.

<https://hu.wikipedia.org/wiki/CSS>

7.

<https://hu.wikipedia.org/wiki/PHP>

8.

[https://hu.wikipedia.org/wiki/Visual Studio Code](https://hu.wikipedia.org/wiki/Visual_Studio_Code)