Űrhajós játék fejlesztése JavaScript és Phaser segítségével

Versenyfelkészítő kézikönyv kezdőknek

Cél: Önálló játékfejlesztés AI segítségével

Tartalomjegyzék

---

0. Fejezet – Indulás: Eszközök és környezet

Mit fogsz megtanulni?

Ebben a fejezetben megtanulod, hogyan állítsd be a professzionális fejlesztői környezetedet. Ez az alapja mindennek – ha ezt jól csinálod, a többi sokkal könnyebb lesz!

0.1 Visual Studio Code – A fejlesztő legjobb barátja

A VSCode olyan, mint egy svájci bicska a programozóknak. Ingyenes, gyors, és rengeteg segítséget ad.

Miért pont VSCode?

- Automatikusan színezi a kódot → könnyebb olvasni
- Hibajelzés gépelés közben → kevesebb bug
- Bővítmények → szuperképességek
- Profi programozók is ezt használják

Telepítés 3 lépésben:

1. **Letöltés:** Menj a https://code.visualstudio.com oldalra
2. **Telepítés:** Futtasd a letöltött fájlt, kattints "Next, Next, Install"
3. **Elindítás:** Nyisd meg a VSCode-ot

Fontos beállítás (opcionális, de hasznos):

Menj a **File → Preferences → Settings** menüpontba, és keresd meg:

- **Auto Save**: állítsd `afterDelay`-re (automatikus mentés)
- **Font Size**: 14-16 közötti (kényelmes olvasás)

0.2 Projekt létrehozása – A HELYES módszer

⚠ **Nagyon fontos:** Soha ne nyiss meg egyetlen fájlt! Mindig az egész projekt mappát nyisd meg!

Miért? Mert a játékod nem egy fájl. Lesznek benne:

- HTML fájlok (a játék szerkezete)
- JavaScript fájlok (a játék logikája)
- Képek (sprites, háttér)
- Hangok (effektek, zene)
- CSS fájlok (stílus)

Lépésről lépésre:

```
1. Nyisd meg a Dokumentumok mappát
```

```
2. Jobb klikk → Új mappa
3. Elnevezés: "urhajo-jatek" (NE használj ékezetes betűt, szóközt!)
4. VSCode-ban: File → Open Folder
5. Válaszd ki az "urhajo-jatek" mappát
6. Kattints: "Select Folder"
```

Helyes mappa struktúra:

```
urhajo-jatek/
├── index.html          (főoldal)
├── main.js             (fő játék kód)
├── assets/             (erőforrások)
│   ├── images/         (képek)
│   ├── sounds/         (hangok)
│   └── fonts/          (betűtípusok)
└── README.md           (dokumentáció)
```

0.3 Az első két fájl – Hello World!

index.html létrehozása

1. VSCode Explorer panel (bal oldalt)
2. Kattints a **New File** ikonra (+)
3. Írd be: `index.html` és nyomj Enter-t

Másold be ezt a kódot:

```html
<!DOCTYPE html>
<html lang="hu">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Űrhajós Kaland - Az én játékom</title>
    <style>
        body {
            margin: 0;
            padding: 0;
            background-color: #000033;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            font-family: Arial, sans-serif;
        }
    </style>
</head>
<body>
    <div id="game-container">
        <h1 style="color: white; text-align: center;"> Űrhajós Kaland </h1>
        <p style="color: #00ff00; text-align: center;">Betöltés...</p>
    </div>

    <!-- A JavaScript fájl betöltése -->
    <script src="main.js"></script>
</body>
</html>
```

Kód magyarázat (sorról sorra):

- `<!DOCTYPE html>` = Megmondja a böngészőnek: ez egy HTML5 oldal
- `<meta charset="UTF-8">` = Magyar ékezetek is működnek
- `<title>` = A böngésző fül címe (fontos a versenyen, mert látszik!)
- `<style>` = CSS kód, ami a kinézetet állítja

- `background-color: #000033` = Sötétkék háttér (űr érzés)
- `display: flex` = Modern elrendezés (középre igazítás)
- `<script src="main.js">` = Itt töltődik be a játék kódja

main.js létrehozása

1. Újra a **New File** ikon
2. Írd be: `main.js` és nyomj Enter-t

Első JavaScript kódod:

```
// Ez egy komment - a gép nem hajtja végre, csak emberi jegyzet

console.log(" A játék motorja elindult!");
console.log("Készítette: [IDE ÍRD A NEVED]");
console.log("Verziószám: 1.0.0");

// Változók a játék állapotának tárolásához
let jatekFut = false;
let pontszam = 0;
let elet = 3;

console.log("Kezdő állapot:");
console.log("- Pontszám:", pontszam);
console.log("- Életek:", elet);
```

Magyarázat:

- `//` = Ez után komment következik (magyarázat)
- `console.log()` = Kiír valamit a böngésző konzoljába (hibakereséshez NAGYON fontos!)
- `let` = Változót hoz létre (olyan, mint egy doboz, amiben adatot tárolsz)

0.4 Live Server – A varázslat kulcsa

Mi az a Live Server?

Egy VSCode bővítmény, ami:

- Automatikusan megnyitja a játékod a böngészőben
- **Azonnal frissíti**, amikor mentesz (nem kell F5-öt nyomni!)
- Professzionális fejlesztői élmény

Telepítés:

1. VSCode bal oldali menü: **Extensions** (4 kocka ikon) VAGY nyomd meg: `Ctrl+Shift+X`
2. Kereső mezőbe írd: **Live Server**
3. Ritwick Dey-től az "Live Server" bővítmény
4. Kattints: **Install**
5. Várd meg, míg települ (néhány másodperc)

Használat:

1. Kattints az `index.html` fájlra (hogy "aktív" legyen)
2. Jobb alsó sarokban megjelenik: **Go Live** gomb
3. Kattints rá!
4. Megnyílik a böngésző: `http://127.0.0.1:5500/index.html`

Sikeres, ha látod:

- A címet: "Űrhajós Kaland - Az én játékom"
- A szöveget: "Betöltés..."

0.5 A böngésző konzol – A fejlesztő röntgenszemüvege

Mit csinál a konzol?

A konzol olyan, mint egy titkos ablak a játékod belsejébe. Itt látod:

- Mit csinál éppen a kód

- Hol van hiba
- Milyen értékek vannak a változókban

Megnyitás:

```
Windows: F12 vagy Ctrl+Shift+I
Mac: Cmd+Option+I
```

Vagy: Jobb klikk az oldalon → **Inspect** → **Console** fül

Mit látsz most?

Ha mindent jól csináltál:

```
 A játék motorja elindult!
Készítette: [A te neved]
Verziószám: 1.0.0
Kezdő állapot:
- Pontszám: 0
- Életek: 3
```

Ha HIBÁT látsz (piros szöveg):

```
Uncaught ReferenceError: xyz is not defined
```

Ez normális! Tanulás közben MINDENKI csinál hibákat. A jó programozó gyorsan megtalálja és kijavítja őket.

Gyakori hibák kezdőknek:

Hibaüzenet	Mit jelent	Megoldás
`not defined`	Olyan változót használsz, ami nem létezik	Ellenőrizd: írtad-e a `let nev = ...` sort?
`Unexpected token`	Szintaktikai hiba (pl. pont vessző hiányzik)	Nézd meg a sor végén van-e `;`
`Cannot read property of undefined`	Valaminek nincs értéke (undefined)	Ellenőrizd, hogy létrehoztad-e a változót

1. Fejezet – JavaScript alapok NULLÁRÓL

Mit fogsz megtanulni?

Ebben a fejezetben a JavaScript alapjait sajátítod el. Ne ijedj meg, ha eleinte furcsának tűnik – minden programozó így kezdte!

1.1 Mi az a JavaScript? (És miért pont ezt tanulod?)

Egyszerű válasz: A JavaScript a weboldalak programozási nyelve.

Részletes válasz:

Képzeld el, hogy egy házat építesz:

- **HTML** = a ház szerkezete (falak, ablak, ajtó)
- **CSS** = a ház kinézete (festés, bútorok, dekoráció)
- **JavaScript** = a ház "intelligenciája" (világítás kapcsoló, ajtócsengő, riasztó)

Miért JavaScript és nem Python/C++/Java?

1. **Böngészőben fut** – nem kell telepíteni semmit
2. **Azonnal látod az eredményt** – módosítasz, frissíted, kész
3. **Játékokhoz tökéletes** – Phaser, PixiJS, Three.js motorok
4. **A világ legelterjedtebb nyelve** – minden weboldal ezt használja

1.2 Változók – Az adatok otthona

Mi az a változó?

Egy "címkézett doboz", amiben valamilyen adatot tárolsz.

Példa a való életből:

```
Képzeld el, hogy van egy dobozod, rá van írva: "játékosNév"
Ebbe a dobozba beleteszed a nevet: "Bence"
Később bármikor megnézheted, vagy kicserélheted: "Márton"
```

JavaScript-ben így néz ki:

```
let jatekosNev = "Bence";
console.log(jatekosNev); // Kiírja: Bence

// Később megváltoztathatod:
jatekosNev = "Márton";
console.log(jatekosNev); // Kiírja: Márton
```

Változó típusok (FONTOS!):

```
// 1. SZÖVEG (String) - idézőjelben
let nev = "Kapitány Kirk";
let uzenet = "Űrhajó sérült!";

// 2. SZÁM (Number) - nincs idézőjel
let pontszam = 0;
let elet = 3;
let sebesseg = 5.5;

// 3. IGAZ/HAMIS (Boolean) - true vagy false
let jatekFut = true;
let jatekVege = false;
let tuzelhet = true;

// 4. TÖMB (Array) - több érték egyszerre
let ellenfelek = ["Meteor", "Idegen hajó", "Űrszemét"];
let pontok = [10, 20, 50, 100];

// 5. OBJEKTUM (Object) - összetett adat
let urhajo = {
    nev: "Enterprise",
    elet: 100,
    pajzs: 50,
    fegyver: "Lézer"
};
```

Játékos példa:

```
// Űrhajó adatai
let urhajoX = 400;        // vízszintes pozíció
let urhajoY = 500;        // függőleges pozíció
let urhajoElet = 100;     // életerő százalékban
let urhajoPajzs = 50;     // pajzs százalékban

// Játék állapot
let jatekPontszam = 0;
let szint = 1;
let ellenfelekSzama = 5;

console.log("Űrhajó indulási adatok:");
console.log("Pozíció X:", urhajoX);
console.log("Pozíció Y:", urhajoY);
console.log("Életerő:", urhajoElet + "%");
```

1.3 Műveletek – Számolás változókkal

Alapvető matematika:

```
let a = 10;
let b = 3;

console.log("Összeadás:", a + b);       // 13
console.log("Kivonás:", a - b);         // 7
console.log("Szorzás:", a * b);         // 30
console.log("Osztás:", a / b);          // 3.3333...
console.log("Maradék:", a % b);         // 1 (10 osztva 3-mal, maradék 1)
```

Játékban használt műveletek:

```
// Pontszám növelése
let pontszam = 0;
pontszam = pontszam + 10;  // Most 10
pontszam = pontszam + 20;  // Most 30

// RÖVIDEBB írásmód (ugyanaz):
pontszam += 10;  // Hozzáad 10-et
pontszam -= 5;   // Levon 5-öt
pontszam *= 2;   // Megszorozza 2-vel
pontszam /= 2;   // Elosztja 2-vel

// Egyel növelés/csökkentés (nagyon gyakori!):
pontszam++;  // Ugyanaz, mint: pontszam = pontszam + 1
elet--;      // Ugyanaz, mint: elet = elet - 1

console.log("Végső pontszám:", pontszam);
```

1.4 Feltételek – Döntéshozatal (if / else)

Miért kell ez?

A játéknak döntéseket kell hoznia:

- Ha az űrhajó kilép a képernyőből → visszahelyezés
- Ha eltalál egy ellenséget → pont hozzáadása
- Ha elfogynak az életek → játék vége

Alap szintaxis:

```
if (feltétel) {
    // Ez fut le, ha a feltétel IGAZ
} else {
    // Ez fut le, ha a feltétel HAMIS
}
```

Konkrét példák:

```
// 1. Egyszerű életellenőrzés
let elet = 0;

if (elet <= 0) {
    console.log("❌ JÁTÉK VÉGE!");
} else {
    console.log("✅ Még élsz, folytatódik a játék!");
}

// 2. Szint nehézség beállítása
let pontszam = 250;
```

```
if (pontszam < 100) {
    console.log("Szint: KÖNNYŰ");
} else if (pontszam < 300) {
    console.log("Szint: KÖZEPES");
} else {
    console.log("Szint: NEHÉZ");
}

// 3. Határellenőrzés (NAGYON gyakori játékokban!)
let urhajoX = 850;
let kepernyoSzelesseg = 800;

if (urhajoX > kepernyoSzelesseg) {
    console.log("⚠  Az űrhajó kilógott a képernyőből!");
    urhajoX = kepernyoSzelesseg; // visszarakjuk
    console.log("✅ Pozíció korrigálva:", urhajoX);
}
```

1.5 Ciklusok – Ismétlés (for loop)

Miért kell ciklus?

Képzeld el, hogy 10 ellenséges űrhajót kell mozgatni. Így is csinálhatnád:

```
//  ROSSZ megoldás (kódismétlés):
ellenseg1X += 1;
ellenseg2X += 1;
ellenseg3X += 1;
// ... és így tovább 10-szer 😱
```

JÓ megoldás ciklussal:

```
for (let i = 0; i < 10; i++) {
    console.log("Ellenseg", i, "mozog...");
}
```

Ciklus felépítése (részletesen):

```
for (kezdőérték; feltétel; léptetés) {
    // Ez ismétlődik, amíg a feltétel igaz
}

// Konkrét példa:
for (let i = 0; i < 5; i++) {
    console.log("Ismétlés száma:", i);
}

// Mit csinál ez?
// 1. i = 0  → konzol: "Ismétlés száma: 0"
// 2. i = 1  → konzol: "Ismétlés száma: 1"
// 3. i = 2  → konzol: "Ismétlés száma: 2"
// 4. i = 3  → konzol: "Ismétlés száma: 3"
// 5. i = 4  → konzol: "Ismétlés száma: 4"
// 6. i = 5  → MEGÁLL (mert 5 NEM < 5)
```

1.6 Függvények – Újrafelhasználható kódrészek

Mi az a függvény?

Olyan, mint egy gép, ami:

1. Kapsz valamilyen **bemenetet** (input)

2. Valami **történik vele**
3. Visszaad egy **eredményt** (output)

Alap szintaxis:

```
function fuggvenyNev(parameter1, parameter2) {
    // Itt történik valami
    return eredmeny;
}

// Meghívás:
let result = fuggvenyNev(10, 20);
```

Konkrét játékbeli példák:

```
// 1. Távolságszámítás két pont között (Pitagorasz-tétel)
function tavolsag(x1, y1, x2, y2) {
    let dx = x2 - x1;
    let dy = y2 - y1;
    let d = Math.sqrt(dx * dx + dy * dy);
    return d;
}

// Használat:
let urhajoX = 100, urhajoY = 100;
let meteorX = 200, meteorY = 200;
let t = tavolsag(urhajoX, urhajoY, meteorX, meteorY);
console.log("Távolság a meteortól:", t, "pixel");

// 2. Véletlen szám generálása tartományban
function veletlen(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

console.log("Véletlen sebesség:", veletlen(1, 10));
console.log("Véletlen X pozíció:", veletlen(0, 800));

// 3. Ellenséges űrhajó létrehozása
function ujEllenseg(x, y, tipus) {
    let ellenseg = {
        x: x,
        y: y,
        tipus: tipus,
        elet: 100,
        sebesseg: veletlen(1, 5)
    };
    console.log("Új ellenség létrehozva:", ellenseg);
    return ellenseg;
}

let ellenseg1 = ujEllenseg(100, 50, "Meteor");
let ellenseg2 = ujEllenseg(300, 80, "Idegen hajó");
```

---

2. Fejezet – A Canvas varázslata

Mit fogsz megtanulni?

A Canvas egy "digitális vászon", amire rajzolhatsz. Ebben a fejezetben megtanulod, hogyan készíts egyszerű animációkat nyers JavaScript-tel. Ez az alapja mindennek, amit a Phaser is csinál a háttérben!

2.1 Mi az a Canvas és miért fontos?

Hasonlat:

Képzeld el, hogy van egy nagy fehér papírod és színes ceruzáid. A Canvas pont így működik, csak digitálisan:

- A papír = `<canvas>` HTML elem
- A ceruzák = JavaScript rajzolási parancsok
- A rajz = a játékod grafikai elemei

Miért tanulod ezt Phaser előtt?

1. **Megérted, mi történik a háttérben** – nem vakon használod a motorot
2. **Hibakereséskor** tudod, mit keresel
3. **Versenypontszám** – ha érted az alapokat, jobban tervezed a játékot

2.2 Canvas létrehozása HTML-ben

index.html módosítása:

Cseréld ki a `<body>` részt erre:

```
<body>
    <div style="text-align: center;">
        <h1 style="color: white;"> Canvas Teszt</h1>
        <canvas id="gameCanvas" width="800" height="600"
                style="border: 2px solid #00ff00; background-color: #000033;">
        </canvas>
        <p style="color: #00ff00;">Canvas méret: 800x600 pixel</p>
    </div>
    <script src="main.js"></script>
</body>
```

2.3 Canvas elérése JavaScript-ből

main.js kód:

```
// 1. CANVAS ELEM ELÉRÉSE
const canvas = document.getElementById("gameCanvas");

// Ellenőrzés: létezik-e?
if (!canvas) {
    console.error("❌ HIBA: Canvas nem található!");
} else {
    console.log("✅ Canvas elem megtalálva:", canvas);
}

// 2. RAJZOLÁSI KONTEXTUS (2D rajzoláshoz)
const ctx = canvas.getContext("2d");

console.log("✅ Rajzolási kontextus létrehozva");
console.log("Canvas szélesség:", canvas.width);
console.log("Canvas magasság:", canvas.height);
```

2.4 Alap formák rajzolása

```
// KITÖLTÖTT TÉGLALAP
ctx.fillStyle = "red";  // Szín beállítása
ctx.fillRect(50, 50, 100, 80);  // x, y, szélesség, magasság

// KÖR RAJZOLÁSA
ctx.beginPath();
ctx.arc(400, 300, 50, 0, 2 * Math.PI);
ctx.fillStyle = "blue";
ctx.fill();
```

```
// SZÖVEG KIÍRÁSA
ctx.font = "30px Arial";
ctx.fillStyle = "white";
ctx.fillText("PONTSZÁM: 0", 50, 500);
```

2.5 Képek betöltése és rajzolása

```
// KÉP OBJEKTUM LÉTREHOZÁSA
const urhajoKep = new Image();

// BETÖLTÉS ELŐTTI ESEMÉNY
urhajoKep.onload = function() {
    console.log("✅ Űrhajó kép betöltve!");
    ctx.drawImage(urhajoKep, 100, 100);
    ctx.drawImage(urhajoKep, 300, 100, 80, 80);  // átméretezve
};

// HIBA KEZELÉSE
urhajoKep.onerror = function() {
    console.error("❌ HIBA: Nem sikerült betölteni a képet!");
};

// FORRÁS BEÁLLÍTÁSA
urhajoKep.src = "urhajo.png";
```

2.6 Kép mozgatása - ANIMÁCIÓ!

```
// GLOBÁLIS VÁLTOZÓK
let urhajoX = 100;
let urhajoY = 300;
let sebesseg = 2;

const urhajoKep = new Image();
urhajoKep.src = "urhajo.png";

urhajoKep.onload = function() {
    jatekLoop();
};

// JÁTÉK LOOP (60 FPS)
function jatekLoop() {
    // 1. TÖRLÉS
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // 2. FRISSÍTÉS
    urhajoX += sebesseg;
    if (urhajoX > canvas.width) {
        urhajoX = -80;
    }

    // 3. RAJZOLÁS
    ctx.drawImage(urhajoKep, urhajoX, urhajoY, 80, 80);

    // 4. KÖVETKEZŐ FRAME
    requestAnimationFrame(jatekLoop);
}
```

---

3. Fejezet – Phaser: A játékmotor titkai

Mit fogsz megtanulni?

Most jön a "varázsrész"! A Phaser egy profi játékmotor, ami automatizálja azt, amit az előbb kézzel csináltál.

3.1 Miért Phaser és nem tiszta Canvas?

Canvas problémái nagy játékoknál:

- Képek betöltésének kezelése
- Sprite-ok pozíciójának tárolása
- Ütközésvizsgálat programozása
- Animációk kezelése (sprite sheet)
- Billentyűzet/egér események
- Hanglejátszás
- Fizika (gravitáció, sebesség)

Phaser előnyei:

- Automatikus kép betöltés
- Sprite létrehozás 1 sorban
- Beépített ütközésvizsgálat
- Animációk egyszerűen

3.2 Phaser telepítése (CDN módszer)

index.html módosítása:

```
<!DOCTYPE html>
<html lang="hu">
<head>
    <meta charset="UTF-8">
    <title> Űrhajós Játék</title>
    <script src="https://cdn.jsdelivr.net/npm/phaser@3.70.0/dist/phaser.js"></script>
</head>
<body>
    <script src="main.js"></script>
</body>
</html>
```

3.3 Phaser alap sablon - RÉSZLETESEN magyarázva

```
// ====================================
// PHASER KONFIGURÁCIÓ
// ====================================
const config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    backgroundColor: '#000033',
    physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 0 },
            debug: false
        }
    },
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};

const game = new Phaser.Game(config);

// ====================================
// GLOBÁLIS VÁLTOZÓK
```

```
// ====================================
let urhajo;
let cursors;
let lovedekek;
let ellenfelek;
let pontszam = 0;
let pontszamSzoveg;

// ====================================
// 1. PRELOAD - Erőforrások betöltése
// ====================================
function preload() {
    console.log(" Erőforrások betöltése...");
    this.load.image('urhajo', 'assets/images/urhajo.png');
    this.load.image('meteor', 'assets/images/meteor.png');
    this.load.image('laser', 'assets/images/laser.png');
    this.load.audio('shoot', 'assets/sounds/laser.mp3');
}

// ====================================
// 2. CREATE - Játék inicializálása
// ====================================
function create() {
    console.log(" Játék létrehozása...");

    // Űrhajó sprite létrehozása
    urhajo = this.physics.add.sprite(400, 500, 'urhajo');
    urhajo.setCollideWorldBounds(true);

    // Billentyűzet kezelés
    cursors = this.input.keyboard.createCursorKeys();

    // Lövedékek csoport
    lovedekek = this.physics.add.group();

    // Ellenségek csoport
    ellenfelek = this.physics.add.group();

    // Pontszám szöveg
    pontszamSzoveg = this.add.text(16, 16, 'Pont: 0', {
        fontSize: '32px',
        fill: '#fff'
    });

    // SPACE billentyű lövéshez
    this.input.keyboard.on('keydown-SPACE', loves, this);

    // Ellenség generálás időzítővel
    this.time.addEvent({
        delay: 2000,
        callback: ujEllenseg,
        callbackScope: this,
        loop: true
    });

    // Ütközés beállítása
    this.physics.add.overlap(lovedekek, ellenfelek, utkozesLovedekEllenseg, null, this);
}

// ====================================
// 3. UPDATE - Minden frame-ben lefut
// ====================================
function update() {
    // Űrhajó mozgatás
```

```
    if (cursors.left.isDown) {
        urhajo.setVelocityX(-300);
    } else if (cursors.right.isDown) {
        urhajo.setVelocityX(300);
    } else {
        urhajo.setVelocityX(0);
    }
}

// ===================================
// FÜGGVÉNYEK
// ===================================
function loves() {
    let lovedek = lovedekek.create(urhajo.x, urhajo.y - 30, 'laser');
    lovedek.setVelocityY(-400);
    this.sound.play('shoot');
}

function ujEllenseg() {
    let x = Phaser.Math.Between(50, 750);
    let ellenseg = ellenfelek.create(x, 0, 'meteor');
    ellenseg.setVelocityY(Phaser.Math.Between(100, 300));
}

function utkozesLovedekEllenseg(lovedek, ellenseg) {
    lovedek.destroy();
    ellenseg.destroy();
    pontszam += 10;
    pontszamSzoveg.setText('Pont: ' + pontszam);
}
```

3.4 Sprite mozgatása billentyűzettel

```
let cursors;

function create() {
    urhajo = this.add.sprite(400, 300, 'urhajo');
    cursors = this.input.keyboard.createCursorKeys();
}

function update() {
    if (cursors.left.isDown) {
        urhajo.x -= 2;
    }
    if (cursors.right.isDown) {
        urhajo.x += 2;
    }
    if (cursors.up.isDown) {
        urhajo.y -= 2;
    }
    if (cursors.down.isDown) {
        urhajo.y += 2;
    }
}
```

3.5 Egér események

```
function create() {
    this.input.on('pointerdown', function (pointer) {
        urhajo.x = pointer.x;
        urhajo.y = pointer.y;
    });
```

```
}
```

3.6 Forgatás

```
function update() {
    urhajo.angle += 1;
}
```

---

4. Fejezet – Életre kelteni a játékot

Mit fogsz megtanulni?

Most az a rész jön, amikor a játékod igazán játszhatóvá válik: pontozás, ütközések, életkezelés!

4.1 Pontozási rendszer

```
let pontszam = 0;
let pontszamSzoveg;

function create() {
    pontszamSzoveg = this.add.text(16, 16, 'Pont: 0', {
        fontSize: '32px',
        fill: '#fff'
    });
}

function noveljPontot(mennyit) {
    pontszam += mennyit;
    pontszamSzoveg.setText('Pont: ' + pontszam);
}
```

4.2 Életkezelés

```
let elet = 3;
let eletIkonok = [];

function create() {
    for (let i = 0; i < 3; i++) {
        let ikon = this.add.image(700 + (i * 35), 30, 'sziv');
        eletIkonok.push(ikon);
    }
}

function vesztettEletet() {
    elet--;
    if (elet >= 0) {
        eletIkonok[elet].setAlpha(0.3);
    }
    if (elet <= 0) {
        jatekVege();
    }
}
```

4.3 Ütközésvizsgálat

```
function create() {
    this.physics.add.overlap(urhajo, ellenfelek, urhajoTalalat, null, this);
    this.physics.add.overlap(lovedekek, ellenfelek, lovedekTalalat, null, this);
}
```

```
function urhajoTalalat(urhajo, ellenseg) {
    ellenseg.destroy();
    vesztettEletet();
    this.cameras.main.shake(200);
}

function lovedekTalalat(lovedek, ellenseg) {
    lovedek.destroy();
    ellenseg.destroy();
    noveljPontot(10);
}
```

4.4 Game Over és Restart

```
function jatekVege() {
    this.physics.pause();

    let gameOverText = this.add.text(400, 300, 'GAME OVER\n\nPont: ' + pontszam, {
        fontSize: '48px',
        fill: '#f00',
        align: 'center'
    });
    gameOverText.setOrigin(0.5);

    this.input.keyboard.once('keydown-ENTER', () => {
        this.scene.restart();
        pontszam = 0;
        elet = 3;
    });
}
```

---

5. Fejezet – AI: A titkos fegyvered

Mit fogsz megtanulni?

Az AI-t OKOSAN kell használni a versenyen. Itt megtanulod hogyan!

5.1 AI asszisztensek a versenyen

Megengedett eszközök:

- ChatGPT (chat.openai.com)
- Claude (claude.ai)
- GitHub Copilot

Szabályok:

- Az AI segíthet ötletelésben, magyarázatban, hibakeresésben
- A végső kódot ÉRTENED KELL
- Dokumentáld, miben segített az AI

5.2 Hogyan kérdezz JÓL?

ROSSZ prompt:

```
"Írj nekem egy űrhajós játékot"
```

JÓ prompt:

```
"Segíts írni egy Phaser 3 függvényt, ami:
- Generál egy véletlen pozícióban meteor sprite-ot
```

```
- A meteor lefelé mozog véletlen sebességgel (100-300 px/s)
- Ha kimegy a képernyőből alul, törlődik"
```

5.3 AI szerepek projektben

1. ÖTLETGENERÁTOR

```
Prompt: "Adj 10 kreatív power-up ötletet egy űrhajós játékhoz"
```

2. KÓDMAGYARÁZÓ

```
Prompt: "Mit csinál ez a kód soronként?
this.physics.add.overlap(bullets, enemies, hitEnemy, null, this);"
```

3. HIBAKERESŐ

```
Prompt: "Debug segítség:
Probléma: A sprite nem jelenik meg
Kód: [ide másold a kódot]
Console: [ide a hibaüzenetet]"
```

5.4 AI-NAPLÓ SABLON

```
## AI Segítség Dokumentáció

### Miben segített az AI?

**1. Ötletelés (dátum)**
- Prompt: "..."
- Eredmény: ...
- Felhasználás: ...

**2. Fizika probléma (dátum)**
- Probléma: ...
- Megoldás: ...

### Mit tanultam?
- ...

### Amit MAGAM oldottam meg:
- ...
```

---

6. Fejezet – Csapatmunka GitHub-bal

Mit fogsz megtanulni?

A 4 fős csapat hogyan dolgozhat egyszerre úgy, hogy ne legyen káosz!

6.1 ARANY SZABÁLY

Soha ne dolgozz közvetlenül a `main` ágon!

6.2 Workflow lépésről lépésre

1. PROJEKT INDÍTÁS (1 fő)

```
GitHub Desktop:
File → New Repository
Name: urhajo-jatek
Initialize with README
```

```
Create Repository
```

2. MUNKAMENET KEZDÉSE (MINDENKI!)

```
1. Fetch origin (frissítés)
2. Current Branch: main
3. Branch → New Branch
4. Name: "sajat-neved-feature"
5. Create Branch
```

3. FEJLESZTÉS

- Kódolás
- Mentés (Ctrl+S)
- Tesztelés

4. COMMIT

```
GitHub Desktop:
Summary: "Mit csináltál?"
Commit to [ágad neve]
```

5. PUSH

```
Publish branch (első alkalom)
vagy
Push origin
```

6. PULL REQUEST

```
Branch → Create Pull Request
GitHub weboldalon:
- Title: "..."
- Description: "..."
- Reviewers: csapattársak
- Create Pull Request
```

7. CODE REVIEW

- Csapattársak nézik át
- Ha jó: Merge
- Ha probléma: módosítás kérése

8. SZINKRONIZÁLÁS (MINDENKI!)

```
1. Válts vissza main ágra
2. Fetch origin
3. Pull origin
```

---

7. Fejezet – Játékfejlesztési stratégiák

MVP (Minimum Viable Product)

Mit KELL tartalmaznia 1. nap végére?

✅ Működő Phaser setup ✅ Űrhajó mozog ✅ Űrhajó tud lőni ✅ 1 fajta ellenség ✅ Ütközés működik ✅ Pontszám számolódik

7.1 Időgazdálkodás

3 napos verseny:

Idő	Feladat

		Kész %
1. nap 12:00	MVP működik	40%
1. nap 17:00	Ellenségek + ütközés	60%
2. nap 12:00	Power-up + hang	75%
2. nap 17:00	Szintek + polish	85%
3. nap 12:00	Bugfix	95%
3. nap 15:00	README, prezentáció	100%

7.2 Feature priority

Prioritás 1 (KELL):

- Több ellenség típus
- Élet rendszer
- Game Over
- Restart
- Szint rendszer

Prioritás 2 (FONTOS):

- Power-upok
- Hangeffektek
- Particle effektek
- High score

Prioritás 3 (NICE TO HAVE):

- Menü
- Több pálya
- Boss
- Teljes történet

---

8. Fejezet – Versenyfelkészítő tippek

8.1 Amit a bírák szeretnek

✅ Egyedi játékmechanika ✅ Smooth animációk ✅ Jó hangok ✅ Particle effektek ✅ Skálázható nehézség ✅ Tiszta kód ✅ Jó README

8.2 Amit kerülj el

❌ Feature creep (100 ötlet, egyik sem működik) ❌ Túl nehéz/könnyű ❌ Rossz optimalizáció (60 FPS alatt) ❌ Bugos állapot ❌ Copyright probléma

8.3 Free asset források

Grafika:

- OpenGameArt.org
- Kenney.nl
- itch.io

Hangok:

- Freesound.org
- ZapSplat.com

Vagy használje erre is AI-t:

- zene -> Suno: https://app.suno.ai/create/
- kép -> DALL·E: https://openai.com/index/dall-e-3/, https://leonardo.ai/
- hang -> ElevenLabs: https://elevenlabs.io/

8.4 Prezentáció

5 perces struktúra:

1. **Intro (30s)** - "Sziasztok! Mi vagyunk..."
2. **Demo (2 perc)** - Játszd le ÉLŐBEN!
3. **Technikai háttér (1.5 perc)** - Phaser 3, GitHub, AI
4. **Challenges (1 perc)** - Legnagyobb kihívás és megoldás
5. **Q&A (30s)** - Kérdések

---

9. Fejezet – CheatSheet (Puska)

JavaScript alapok

```
// ===== VÁLTOZÓK =====
let x = 10;                    // Változtatható
const ÁLLANDÓ = 100;           // Nem változtatható
var regi = 5;                  // NE HASZNÁLD (régi stílus)

// ===== ADATTÍPUSOK =====
let szoveg = "Hello";
let szam = 42;
let logikai = true;
let tomb = [1, 2, 3, 4, 5];
let objektum = { x: 10, y: 20 };
let ures = null;
let nincs = undefined;

// ===== TÖMB MŰVELETEK =====
let t = [10, 20, 30];
t.push(40);                    // Hozzáad a végére → [10,20,30,40]
t.pop();                       // Törli az utolsót → [10,20,30]
t.shift();                     // Törli az elsőt → [20,30]
t.unshift(5);                  // Hozzáad elejére → [5,20,30]
t.length;                      // Elemek száma → 3
t[0];                          // Első elem → 5
t.forEach(x => console.log(x));// Végigmegy mindenen

// ===== OBJEKTUM MŰVELETEK =====
let obj = { nev: "Űrhajó", elet: 100 };
obj.nev;                       // "Űrhajó"
obj["nev"];                    // "Űrhajó" (alternatív)
obj.pajzs = 50;                // Új property hozzáadása
delete obj.pajzs;              // Property törlése

// ===== FELTÉTELEK =====
if (x > 5) {
    console.log("Nagy");
} else if (x > 0) {
    console.log("Kicsi");
} else {
    console.log("Nulla vagy negatív");
}

// Ternary operator (rövid if)
let eredmeny = (x > 5) ? "Nagy" : "Kicsi";

// Switch
switch(x) {
    case 1: console.log("Egy"); break;
    case 2: console.log("Kettő"); break;
    default: console.log("Más");
}

// ===== CIKLUSOK =====
for (let i = 0; i < 10; i++) { }          // Klasszikus for
for (let elem of tomb) { }                // Tömb elemein végigmegy
```

```
for (let index in tomb) { }                // Indexeken megy végig
tomb.forEach(elem => { });                 // Modern forEach

while (x < 10) { x++; }                    // While ciklus
do { x++; } while (x < 10);                // Do-while (min 1x lefut)

// ===== FÜGGVÉNYEK =====
function osszeadas(a, b) {
    return a + b;
}

// Arrow function (modern)
const szorzas = (a, b) => a * b;
const negyzet = x => x * x;                // 1 paraméter, nem kell ()

// Callback függvény
function csinalValamit(callback) {
    callback();
}
csinalValamit(() => console.log("Kész!"));

// ===== STRING MŰVELETEK =====
let s = "Hello World";
s.length;                       // 11
s.toUpperCase();               // "HELLO WORLD"
s.toLowerCase();               // "hello world"
s.includes("World");           // true
s.split(" ");                  // ["Hello", "World"]
s.substring(0, 5);             // "Hello"
s.replace("World", "JS");      // "Hello JS"

// Template literal (backtick)
let nev = "Bence";
let uzenet = `Szia ${nev}!`;  // "Szia Bence!"

// ===== MATEMATIKA =====
Math.floor(4.7);               // 4 (lefelé kerekít)
Math.ceil(4.3);                // 5 (felfelé kerekít)
Math.round(4.5);               // 5 (normál kerekítés)
Math.abs(-5);                  // 5 (abszolút érték)
Math.max(1, 5, 3);             // 5
Math.min(1, 5, 3);             // 1
Math.pow(2, 3);                // 8 (2^3)
Math.sqrt(16);                 // 4 (négyzetgyök)
Math.random();                 // 0 és 1 között véletlen
Math.PI;                       // 3.14159...
```

Phaser 3 - Teljes referencia

```
// ===== KONFIGURÁCIÓ =====
const config = {
    type: Phaser.AUTO,              // WebGL vagy Canvas
    width: 800,
    height: 600,
    backgroundColor: '#000033',
    parent: 'game-container',       // HTML elem ID
    physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 0 },
            debug: false            // true = láthatók a fizikai testek
        }
```

```
    },
    scale: {
        mode: Phaser.Scale.FIT,     // Automatikus skálázás
        autoCenter: Phaser.Scale.CENTER_BOTH
    },
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};

const game = new Phaser.Game(config);

// ===== PRELOAD - Fájlok betöltése =====
function preload() {
    // Képek
    this.load.image('key', 'path/to/image.png');
    this.load.image('bg', 'assets/background.jpg');

    // Sprite sheet (animációhoz)
    this.load.spritesheet('player', 'player.png', {
        frameWidth: 32,
        frameHeight: 48
    });

    // Hangok
    this.load.audio('music', 'music.mp3');
    this.load.audio('sfx', ['sound.mp3', 'sound.ogg']); // Több formátum

    // JSON
    this.load.json('level', 'level1.json');

    // Betöltési progress bar
    this.load.on('progress', (value) => {
        console.log('Betöltés:', Math.round(value * 100) + '%');
    });
}

// ===== CREATE - Játék inicializálása =====
function create() {
    // ----- KÉPEK ÉS SPRITE-OK -----
    this.add.image(400, 300, 'bg');                    // Kép (nem mozog)
    let sprite = this.add.sprite(100, 100, 'player');  // Sprite

    // Fizikai sprite (ütközhet, mozoghat)
    let fizikaiSprite = this.physics.add.sprite(200, 200, 'enemy');
    fizikaiSprite.setCollideWorldBounds(true);         // Ne menjen ki a képből
    fizikaiSprite.setBounce(0.5);                      // Pattanás
    fizikaiSprite.setVelocity(100, 200);               // Sebesség
    fizikaiSprite.setScale(2);                         // Dupla méret
    fizikaiSprite.setAlpha(0.5);                       // 50% átlátszó
    fizikaiSprite.setTint(0xff0000);                   // Piros színezés
    fizikaiSprite.setDepth(10);                        // Z-index (réteg)

    // ----- CSOPORTOK -----
    let csoport = this.physics.add.group({
        defaultKey: 'enemy',
        maxSize: 50,
        collideWorldBounds: true
    });

    // Elemek hozzáadása
    for (let i = 0; i < 10; i++) {
```

```
        let elem = csoport.create(i * 50, 100, 'enemy');
    }

    // ----- SZÖVEG -----
    let szoveg = this.add.text(16, 16, 'Pont: 0', {
        fontSize: '32px',
        fill: '#ffffff',
        fontFamily: 'Arial',
        backgroundColor: '#000000',
        padding: { x: 10, y: 5 },
        align: 'center'
    });
    szoveg.setOrigin(0.5);                          // Középre igazítás
    szoveg.setShadow(2, 2, '#000000', 2);           // Árnyék

    // ----- ANIMÁCIÓ -----
    this.anims.create({
        key: 'walk',
        frames: this.anims.generateFrameNumbers('player', {
            start: 0,
            end: 3
        }),
        frameRate: 10,
        repeat: -1                                  // Végtelen ismétlés
    });
    sprite.anims.play('walk');

    // ----- BILLENTYŰZET -----
    this.cursors = this.input.keyboard.createCursorKeys();

    // Egyedi billentyűk
    this.spaceKey = this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.SPACE);
    this.wKey = this.input.keyboard.addKey('W');

    // Billentyű esemény
    this.input.keyboard.on('keydown-SPACE', () => {
        console.log('SPACE megnyomva!');
    });

    // ----- EGÉR / ÉRINTÉS -----
    this.input.on('pointerdown', (pointer) => {
        console.log('Kattintás:', pointer.x, pointer.y);
    });

    this.input.on('pointermove', (pointer) => {
        sprite.x = pointer.x;
        sprite.y = pointer.y;
    });

    // ----- IDŐZÍTŐK -----
    // Egyszeri esemény (3 mp múlva)
    this.time.delayedCall(3000, () => {
        console.log('3 másodperc eltelt!');
    });

    // Ismétlődő esemény
    this.time.addEvent({
        delay: 1000,                                // 1 másodperc
        callback: () => {
            console.log('Tick!');
        },
        loop: true
    });
```

```
    // Timer objektum (megállítható)
    this.timer = this.time.addEvent({
        delay: 500,
        callback: this.enemyShoot,
        callbackScope: this,
        loop: true
    });
    // Megállítás: this.timer.remove();

    // ----- ÜTKÖZÉSEK -----
    // Overlap (átfedés, nem állítja meg az objektumokat)
    this.physics.add.overlap(player, coins, collectCoin, null, this);

    // Collider (valódi ütközés, megállítja az objektumokat)
    this.physics.add.collider(player, platforms);

    // Csoport vs csoport
    this.physics.add.overlap(bullets, enemies, hitEnemy, null, this);

    // ----- KAMERA -----
    this.cameras.main.setBounds(0, 0, 1600, 1200);    // Világ mérete
    this.cameras.main.startFollow(player);             // Követ egy sprite-ot
    this.cameras.main.setZoom(1.5);                    // Zoom
    this.cameras.main.shake(200, 0.01);                // Rázkódás effekt
    this.cameras.main.flash(250);                      // Villanás
    this.cameras.main.fade(1000);                      // Kifakulás

    // ----- HANGOK -----
    this.sound.play('sfx');                            // Egyszeri hang

    let music = this.sound.add('music', {
        loop: true,
        volume: 0.5
    });
    music.play();
    // music.pause();
    // music.resume();
    // music.stop();

    // ----- TWEENS (ANIMÁCIÓK) -----
    this.tweens.add({
        targets: sprite,
        x: 700,                                        // Célpozíció
        duration: 2000,                                // 2 másodperc
        ease: 'Power2',                                // Easing
        yoyo: true,                                    // Vissza is megy
        repeat: -1                                     // Végtelen
    });

    // Több property egyszerre
    this.tweens.add({
        targets: sprite,
        alpha: 0,
        scale: 2,
        angle: 360,
        duration: 1000
    });

    // ----- PARTICLE EFFEKTEK -----
    let particles = this.add.particles('particle');

    let emitter = particles.createEmitter({
        x: 400,
        y: 300,
```

```
        speed: { min: 100, max: 300 },
        angle: { min: 0, max: 360 },
        scale: { start: 1, end: 0 },
        lifespan: 1000,
        quantity: 2,
        blendMode: 'ADD'
    });

    // Sprite-hoz csatolás
    emitter.startFollow(player);
}

// ===== UPDATE - Minden frame =====
function update(time, delta) {
    // time = játék kezdete óta eltelt idő (ms)
    // delta = előző frame óta eltelt idő (ms)

    // Billentyűzet kezelés
    if (this.cursors.left.isDown) {
        player.setVelocityX(-200);
    } else if (this.cursors.right.isDown) {
        player.setVelocityX(200);
    } else {
        player.setVelocityX(0);
    }

    // Ugrás
    if (this.cursors.up.isDown && player.body.touching.down) {
        player.setVelocityY(-400);
    }

    // Sprite körbeforgatása
    player.angle += 1;

    // Távolság ellenőrzés
    let dist = Phaser.Math.Distance.Between(
        player.x, player.y,
        enemy.x, enemy.y
    );
    if (dist < 50) {
        console.log('Közel van!');
    }
}

// ===== HASZNOS CALLBACK FÜGGVÉNYEK =====
function collectCoin(player, coin) {
    coin.destroy();                                    // Coin eltűnik
    score += 10;
    scoreText.setText('Pont: ' + score);
}

function hitEnemy(bullet, enemy) {
    bullet.destroy();
    enemy.destroy();

    // Robbanás effekt
    let explosion = this.add.sprite(enemy.x, enemy.y, 'explosion');
    explosion.anims.play('explode');
    explosion.on('animationcomplete', () => {
        explosion.destroy();
    });
}
```

Gyakran használt Math függvények játékokhoz

```
// ===== VÉLETLEN SZÁMOK =====
// Véletlen 0 és 1 között
Math.random();                                          // pl. 0.634

// Véletlen egész szám min és max között (inclusiv)
function randomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
randomInt(1, 10);                                       // pl. 7

// Phaser beépített
Phaser.Math.Between(1, 10);                             // 1-10 között
Phaser.Math.FloatBetween(1.5, 5.5);                     // Tört is lehet

// Véletlen elem tömbből
let colors = ['red', 'blue', 'green'];
Phaser.Utils.Array.GetRandom(colors);                   // pl. 'blue'

// ===== TÁVOLSÁG ÉS SZÖG =====
// Távolság két pont között (Pitagorasz)
let dist = Math.sqrt(
    Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2)
);

// Phaser beépített
Phaser.Math.Distance.Between(x1, y1, x2, y2);

// Szög két pont között (radiánban)
let angle = Math.atan2(y2 - y1, x2 - x1);

// Phaser beépített (radiánban)
Phaser.Math.Angle.Between(x1, y1, x2, y2);

// Radián <-> Fok átváltás
let radianToFok = radián * (180 / Math.PI);
let fokToRadian = fok * (Math.PI / 180);

// Phaser beépített
Phaser.Math.RadToDeg(radián);
Phaser.Math.DegToRad(fok);

// ===== INTERPOLÁCIÓ (LERP) =====
// Két érték között simán átmegy
function lerp(start, end, t) {
    return start + (end - start) * t;
}
// t = 0 → start
// t = 0.5 → középen
// t = 1 → end

// Phaser beépített
Phaser.Math.Linear(start, end, t);

// ===== CLAMP (KORLÁTOZÁS) =====
// Érték korlátok közé szorítása
function clamp(value, min, max) {
    return Math.max(min, Math.min(max, value));
}
clamp(150, 0, 100);                                     // 100

// Phaser beépített
Phaser.Math.Clamp(value, min, max);
```

```
// ===== KEREKÍTÉS =====
Math.floor(4.7);                                    // 4 (lefelé)
Math.ceil(4.3);                                     // 5 (felfelé)
Math.round(4.5);                                    // 5 (közelebb)
(4.567).toFixed(2);                                 // "4.57" (2 tizedesjegy)

// ===== NORMALIZÁLÁS =====
// Vektor normalizálása (hossz = 1)
function normalize(x, y) {
    let length = Math.sqrt(x * x + y * y);
    return {
        x: x / length,
        y: y / length
    };
}
```

Design Pattern-ek játékokhoz

```
// ===== OBJECT POOL (újrahasznosítás) =====
// Jobb performancia, kevesebb szemétgyűjtés
function create() {
    this.bulletPool = this.physics.add.group({
        defaultKey: 'bullet',
        maxSize: 50,
        runChildUpdate: true
    });
}

function fireBullet() {
    let bullet = this.bulletPool.get(player.x, player.y);
    if (bullet) {
        bullet.setActive(true);
        bullet.setVisible(true);
        bullet.setVelocityY(-400);
    }
}

function update() {
    this.bulletPool.children.entries.forEach(bullet => {
        if (bullet.y < 0) {
            bullet.setActive(false);
            bullet.setVisible(false);
        }
    });
}

// ===== STATE MACHINE (állapotgép) =====
let enemyState = {
    IDLE: 'idle',
    PATROL: 'patrol',
    CHASE: 'chase',
    ATTACK: 'attack'
};

let currentState = enemyState.IDLE;

function updateEnemy() {
    switch(currentState) {
        case enemyState.IDLE:
            // Áll
            if (playerNear) currentState = enemyState.CHASE;
```

```
            break;
        case enemyState.CHASE:
            // Üldöz
            moveTowardsPlayer();
            if (playerVeryClose) currentState = enemyState.ATTACK;
            if (!playerNear) currentState = enemyState.IDLE;
            break;
        case enemyState.ATTACK:
            // Támad
            attackPlayer();
            if (!playerVeryClose) currentState = enemyState.CHASE;
            break;
    }
}

// ===== EVENT EMITTER (eseménykezelés) =====
// Globális eseménykezelő
const GameEvents = new Phaser.Events.EventEmitter();

// Feliratkozás eseményre
GameEvents.on('enemyKilled', (points) => {
    score += points;
    updateScore();
});

// Esemény kiváltása
function killEnemy() {
    enemy.destroy();
    GameEvents.emit('enemyKilled', 100);
}

// ===== SINGLETON (egyetlen példány) =====
let GameManager = {
    score: 0,
    lives: 3,
    level: 1,

    addScore(points) {
        this.score += points;
    },

    loseLife() {
        this.lives--;
        if (this.lives <= 0) {
            this.gameOver();
        }
    },

    nextLevel() {
        this.level++;
    },

    gameOver() {
        console.log('Game Over! Score:', this.score);
    }
};
```

Optimalizálási tippek

```
// ===== OFFSCREEN CULLING (nem látható törlése) =====
function update() {
    enemies.children.entries.forEach(enemy => {
```

```
        if (enemy.y > 700 || enemy.y < -100 ||
            enemy.x > 900 || enemy.x < -100) {
            enemy.destroy();
        }
    });
}

// ===== FPS MONITOR =====
function create() {
    this.fpsText = this.add.text(10, 10, '', { fill: '#0f0' });
}

function update() {
    let fps = Math.round(this.game.loop.actualFps);
    this.fpsText.setText('FPS: ' + fps);
}

// ===== TEXTURE ATLAS (sprite sheet optimalizálás) =====
// Preload-ban
this.load.atlas('sprites', 'sprites.png', 'sprites.json');

// Create-ben
this.add.image(100, 100, 'sprites', 'player.png');

// ===== OBJECT REUSE (ne hozz létre újat mindig) =====
// ROSSZ
function shoot() {
    let bullet = this.physics.add.sprite(x, y, 'bullet');
}

// JÓ
let bulletPool = [];
function shoot() {
    let bullet = bulletPool.find(b => !b.active);
    if (bullet) {
        bullet.setPosition(x, y);
        bullet.setActive(true);
    }
}
```

Debug tippek

```
// ===== CONSOLE LOGGING =====
console.log('Érték:', valtozo);
console.warn('Figyelmeztetés!');
console.error('Hiba történt!');
console.table({ x: 10, y: 20, z: 30 });               // Táblázat

// ===== ASSERT =====
console.assert(x > 0, 'X-nek pozitívnak kell lennie!');

// ===== PHASER DEBUG =====
// Config-ban
physics: {
    arcade: {
        debug: true                                    // Fizikai testek láthatóak
    }
}

// Debug szöveg
this.add.text(10, 50, '', { fill: '#0f0' })
    .setScrollFactor(0)                               // Fix pozíció (kamera mozog)
```

```
    .setDepth(1000);                                    // Mindig felül

function update() {
    debugText.setText([
        'Player X: ' + player.x,
        'Player Y: ' + player.y,
        'Velocity: ' + player.body.velocity.x,
        'FPS: ' + Math.round(this.game.loop.actualFps)
    ]);
}

// ===== BREAKPOINT (böngésző DevTools) =====
debugger;                                               // Megállítja a kódot itt
```

ZÁRÓ GONDOLATOK

Utolsó ellenőrző lista leadás előtt:

```
□ Játék elindul hiba nélkül
□ Legalább 3 perc játékidő
□ Game Over működik
□ Újraindítás működik
□ README.md kitöltve
□ Nincs console.error
□ 60 FPS
□ 3 emberrel tesztelve
□ GitHub repo rendezett
□ Prezentáció begyakorolva
```

HAJRÁ! JÓ VERSENYZÉST!