Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Általános Informatikai Intézeti Tanszék



**HTML5 Canvas alapú Space invaders játék fejlesztése webes környezetben**

**Szakdolgozat**

**Készítette:** Nagy Bence  
**Neptunkód:** WH8L7E  
**Szak:** Mérnökinformatikus BSc

Informatikai rendszermérnök szakirány

|  |  |  |  |
| --- | --- | --- | --- |
| **Miskolci Egyetem**  **Gépészmérnöki és Informatikai Kar** |  | | **Általános Informatikai Intézeti Tanszék**  3515 Miskolc-Egyetemváros |
| Szak**: Mérnökinformatikus** | |  | Szakdolgozat azonosító: **IAL/WH8L7E/BSc/2024** |
| Szakirány: Informatikai rendszermérnök | |  | **Intézmény azonosító: FI 87515** |
|  | |  |  |

**SZAKDOLGOZAT FELADAT**

**Nagy BEnce**

BSc mérnökinformatikus jelölt részére

|  |  |
| --- | --- |
| A tervezés tárgyköre: | **Számítógépes játékfejlesztés** |
| A feladat címe: | **HTML5 Canvas alapú Space invaders játék fejlesztése webes környezetben** |
| **A feladat részletezése:**   * A hallgató feladata, hogy irodalomkutatást végezzen a számítógépes játékpiac jelenlegi helyzetéről, vizsgálva a fejlődési irányokat és piaci tendenciákat. * A játékfejlesztési technológiák és motorok általános bemutatása, különös figyelemmel a HTML5 játékfejlesztési lehetőségekre és a canvas technológia szerepére a böngészőalapú fejlesztésben. * Egy „Space Invaders” játék tervezése, amely a különböző pályák, nehézségi szintek és a lebilincselő játékélmény kidolgozását is magába foglalja. * A játékban használt adattárolási megoldás bemutatása, amely megvalósítja az adatok hosszútávú tárolását és visszakereshetőségét webes környezetben. * A játék funkcióit bemutató prototípus alkalmazás készítése, amely a játék főbb elemeit, egyéb menüpontjait, az animációt és felhasználói interakciót szemlélteti. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Tervezésvezető:  Dr. Mileff Péter | | Tanszék, beosztás:  Általános Informatikai Intézeti Tanszék, egyetemi docens | |
| Konzulens(ek):  - | | Cég, beosztás:  - | |
| A szakdolgozat kiadásának időpontja: | 2024.09.09. | |
| A szakdolgozat beadásának határideje: | 2024.11.15. | |
| Miskolc, 2024.11.15. | **Prof. Dr. Kovács László**  tanszékvezető egyetemi tanár | |

1. A szakmai gyakorlat helye: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. A szakmai gyakorlat vezetőjének neve: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. A szakdolgozat módosítása: szükséges (a módosítást külön lap tartalmazza)   
 nem szükséges (a megfelelő rész aláhúzandó)

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

tervezésvezető aláírása

4. A tervezést ellenőriztem: (1) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(3) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(4) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dátum, tervezésvezető aláírása

5. A szakdolgozat beadható

nem adható be

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

konzulens aláírása tervezésvezető aláírása

6. A szakdolgozat ….. szövegoldalt,

….. db rajzot,

….. db CD mellékletet

….. egyéb mellékletet tartalmaz.

7. A szakdolgozat bírálatra: bocsátható

nem bocsátható

A bíráló neve, címe: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

tanszékvezető aláírása

8. Osztályzat: a bíráló javaslata: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a tanszék javaslata: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a Záróvizsga Bizottság döntése: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

a Záróvizsga Bizottság elnökének aláírása

**EREDETISÉGI NYILATKOZAT**

Alulírott Nagy Bence; Neptun-kód: WH8L7E a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős mérnökinformatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy HTML5 Canvas alapú Space Invaders játék fejlesztése webes környezetben című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

* szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
* tartalmi idézet hivatkozás megjelölése nélkül;
* más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy

plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc,.............év ………………..hó ………..nap

…….……………………………….…

Hallgató

Tartalom

[1. Bevezetés 7](#_Toc181397624)

[1.1 Témaválasztás és jelentőség 7](#_Toc181397625)

[1.2 Célkitűzések 7](#_Toc181397626)

[1.3 Motiváció 8](#_Toc181397627)

[2. A játékfejlesztés története 9](#_Toc181397628)

[3. A játékpiac helyzete 11](#_Toc181397629)

[3.1 Történelmi áttekintés 11](#_Toc181397630)

[3.2 A piac hajtóereje 11](#_Toc181397631)

[3.3 A piac eloszlása 13](#_Toc181397632)

[4. A játékfejlesztésnél használt alapfogalmak 16](#_Toc181397633)

[5. Játékfejlesztési motorok 17](#_Toc181397634)

[6. HTML5 játékfejlesztési technológiák 20](#_Toc181397635)

[6.1 Ismertebb keretrendszerek és könyvtárak 20](#_Toc181397636)

[6.2 HTML grafika: CANVAS, SVG 21](#_Toc181397637)

[6.3 Modern webes adattárolási megoldások 23](#_Toc181397638)

[7. Digitális Tartalmak Felhasználása: Licencjogok és Fair Use 27](#_Toc181397639)

[7.1 Általános áttekintés 27](#_Toc181397640)

[7.2 Creative Commons Licenc típusok 28](#_Toc181397641)

[8. Az elkészített program bemutatása 30](#_Toc181397642)

[8.1 A játék eredete 30](#_Toc181397643)

[8.2 A játék rövid bemutatása 30](#_Toc181397644)

[8.3 Programozási környezet ismertetése 31](#_Toc181397645)

[8.4 A játék struktúrája 32](#_Toc181397646)

[8.5 Képek kirajzolása 33](#_Toc181397647)

[8.5.1 Egyszerűbb alakzatok, gombok rajzolása 33](#_Toc181397648)

[8.5.2 Képek kirajzolása a játékban (enemy, player, boss) 34](#_Toc181397649)

[8.5.3 Animáció 36](#_Toc181397650)

[8.6 Játéklogika 37](#_Toc181397651)

[8.7 Canvas eseménykezelés 47](#_Toc181397652)

[8.7.1 Billentyűlenyomás 47](#_Toc181397653)

[8.7.2 Kattintás 47](#_Toc181397654)

[8.7.3 Egérmozgás 49](#_Toc181397655)

[8.8 Adatok tárolása 50](#_Toc181397656)

[8.9 Játékbeli fizetőeszköz használata 51](#_Toc181397657)

[9. Összefoglalás 53](#_Toc181397658)

[10. Summary 54](#_Toc181397659)

[11. Irodalomjegyzék 55](#_Toc181397660)

[12. Mellékletek: 56](#_Toc181397661)

# 1. Bevezetés

## 1.1 Témaválasztás és jelentőség

A játékfejlesztés, mint téma rendkívül fontos a modern digitális kultúrában. A játéktervezés és -fejlesztés amellett, hogy a játékipar egyik legaktívabban fejlődő szegmense, mély elméleti és gyakorlati jelentőséggel is bír. Az elmúlt években a technológiai fejlődése, különösen az internet és a mobil eszközök terjedése, forradalmasította a játékipart is.

Az alábbiakban néhány érvet sorolok fel, amelyek alátámasztják a témaválasztásom relevanciáját:

* kreativitás: A fejlesztőnek ki kell találni új mechanikákat, történeteket, grafikai elemeket, amelyhez nem árt a produktivitás.
* készségfejlesztés és csapatmunka: A programozóknak a játék készítése közben meg kell tanulniuk kódolni, miközben a problémamegoldási képességük fejlődik, és hatékonyan együtt kell működniük a csapattársaikkal.
* szórakoztatás: Alapvetően a játékokat az emberek szórakoztatására, idejének kellemes eltöltése miatt találták ki.
* oktatás: A játékok akár hatékony eszközei lehetnek a tanulásnak. Például lehet velük tanítani matematikát, idegen nyelvet és történelmet is.
* gazdasági lehetőségek: A játékfejlesztés egyre nagyobb piacra tesz szert. Egy sikeres játékkal akár milliókat, még sikeresebbel milliárdokat lehet keresni.

Az elmúlt években az internetes technológiák fejlődése lehetővé tette olyan magas színvonalú játékok létrehozását, amelyeket korábban csak asztali vagy konzolos platformokon lehetett elérni. Az HTML5 alapú játékok könnyen elérhetők és játszhatók bármely modern böngészőben, ami hatalmas piaci lehetőségeket rejthet magában.

## 1.2 Célkitűzések

A játékfejlesztéssel kapcsolatos céljaim közé tartozik a személyes fejlődés, a kreativitás kibontakozása és a játékosok szórakoztatása. Saját játék létrehozásával új készségeket és ismereteket szeretnék elsajátítani a tervezéstől kezdve egészen a programozásig, és felfedezni az alkotás kihívásait.

* **Széles körű megértés:** A projekt fő célja a HTML5 játékfejlesztési folyamatának és technológiájának mélyebb megértése. Ennek érdekében kiterjedt kutatást folytatok a HTML, CSS és JavaScript elemek játékfejlesztésben való felhasználásáról. Ezenkívül részletesen kutattam azokat a konkrét könyvtárakat és keretrendszereket, amelyek lehetővé teszik a játékfejlesztést, hogy átfogó képet kaphassanak ezek működéséről és használatáról.
* **Kreativitás kibontakozása:** A másik lényeges cél a kreativitásom fejlesztése volt a játékfejlesztés során. Célom, hogy kihasználjam a HTML5 adta lehetőségeket, hogy egyedi és innovatív játékélményt alkossak. Ez magában foglalja az animáció, a grafika, a kép és a hang kreatív felhasználását olyan játék létrehozásához, amely leköti a játékosokat, és egyedi játékélményt nyújt.
* **A felhasználói élmény javítása:** A játékosok elégedettsége és szórakoztatása is lényeges szempont a projekt során. Ezért kiemelt figyelmet fordítok a felhasználói élmény folyamatos javítására, fejlesztésére. Céljaim közé tartozik, hogy a játék zökkenőmentesen fusson és izgalmakkal teli játékélményt nyújtson, miközben intuitív és könnyen használható felületet biztosítok a felhasználóknak.

## 1.3 Motiváció

Személyes motivációm videójátékok létrehozására több tényezőből fakad. Először is az önfejlesztés vágya motivál. Játékok készítése közben számos új technológiát és programozási nyelvet tanulok, miközben olyan összetett problémákat oldok meg, amelyek javítják technikai és rálátási készségeimet. Sőt, a kreatív megnyilvánulás lehetősége is erős indíttatást jelent számomra. Saját ötleteim és elképzeléseim életre keltése izgalmas és hasznos tevékenység, amely során kamatoztathatom alkotói és tervezői képességeimet. Emellett a játékfejlesztés célja, hogy szórakoztató és élvezetes élményt nyújtson a játékosoknak. Az a tudat, hogy mások is jól szórakozhatnak az általam készített játékon, erőteljes motiváció a fejlesztési folyamat során. Emellett lelkesít az iparág folyamatos fejlődése és a karrierlehetőségek sokszínűsége, mert a játékfejlesztés területén megszerzett tapasztalatok és készségek értékesek lehetnek a munkaerőpiacon számos területen a játékfejlesztésen kívül is. A kihívások is alapvető ösztönzést jelentenek számomra. A komplex problémák megoldása, a technológiai akadályok leküzdése folyamatosan serkent a tanulásra és a fejlődésre. Továbbá, a kreatív és innovatív megoldások kidolgozása izgalmas és hálás tevékenység, amely lehetőséget kínál arra, hogy a feladat során még tovább fejlesszem képességeimet.

# 2. A játékfejlesztés története

Az emberek évezredek óta próbálnak szórakoztató és unaloműző tevékenységeket találni. Az egyik legelterjedtebb és legrégebbi módszer erre a játék. A múltban a játékkészítés egyszerű kézzel készített játékokkal kezdődött, és az évszázadok során a technológiai és kulturális változásoknak köszönhetően fejlődött. *[1]*

A játékfejlesztés történetének korai szakaszában az emberek elég primitív játékokat hoztak létre, például fafaragásokat, homok- és kavicsos játékokat vagy társasjátékokat. Ezek a játékok gyakran kézzel, egyszerű anyagokból készültek. (például: fák, kövek, agyag, termések, növények, szarvak és agyarak) Ezek nagyon népszerűek, és gyakran használatosak voltak társadalmi, vallási vagy oktatási célokra. Továbbá, jellemző volt még, hogy a család tagjai, ismerősök vagy éppen saját maguknak készítettek a játékokat az emberek. *[1]*

A játékfejlesztés igazi áttörése az ipari forradalommal és a technológia fellendülésével kezdődött. A 20. század elején megjelentek az első mechanikus játékok, majd később az elektromos játékok és a flippergépek. Ezzel együtt népszerűségük is kezdett egyre elterjedtebbé válni a világon. A játékok anyagát tekintve a természetben is könnyen megtalálható anyagokat felváltották a műanyagok, a fémek és egyéb szintetikus anyagok. *[1]*

Az első digitális számítógépek felbukkanásával, új lehetőségek nyíltak meg a játékok létrehozására. Az 1958-ban kifejlesztett „Tennis for Two” például az egyik első digitális számítógépes játék volt, amely egy kezdetleges teniszszimulációt nyújtott a játékosoknak. *[2]*

Az 1970-es években debütáltak az első videojátékok, mint például a „Pong”, amelyeket már a szórakoztató elektronikai iparágban dolgozó vállalatok által jöttek létre. A videojátékok az 1980-as években és az 1990-es évek elején robbanásszerűen terjedtek, az első otthoni konzolok és személyi számítógépek hozzájárultak, hogy az emberek otthon is élvezhessék a játékokat. *[2]*

Az internet térnyerésével és a mobiltechnológia fejlődésével az elmúlt években újabb nagy változások történtek a játékfejlesztés területén. Nem kellett sokat várni a játékforgalmazó platformok megjelenésére sem. Ezeken keresztül elérhetők, letölthetők (ha szükséges) és játszhatók a videójátékok, beleértve az asztali számítógépeket, a konzolokat, a mobiltelefonokat és a böngészőket. *[2]*

Említésként néhány népszerű játékértékesítő platform:

* Asztali pc játékok: Origin, Epic Games Store, Ubisoft, Electronic Arts App, Itch, Steam (asztali számítógép mellett saját fejlesztésű kézikonzolra is készít játékot)
* Konzol: Xbox, Playstation, Nintendo (kézikonzol is), Asus (csak kézikonzol)
* Mobiltelefon: App Store (MacOS), Google Play Store (Android), AppGalery(Huawei), Galaxy Store (Samsung) *[3]*

Ezeken is túlhaladva megjelentek az előfizetős platformok, amiknek főbb lényege: hogy legalább 1 vagy több hónapra fizet a felhasználó a felület használatáért. Ennek előnye: jelentősen kevesebb összeget kell kiadnia a vásárlónak mintha megvenné a játékokat. A vállalat szempontjából pedig: havonta, tehát folyamatosan szerez bevételt. Ilyen platform az: Microsoft Xbox Game Pass szolgáltatása, ami asztali számítógépre és konzolra is elérhető. Továbbá a Playstation Plus szolgáltatása is elérhető a felhasználók számára jelenleg csak és kizárólag konzolra. Ezen felül a Netflix is elkezdte játékok bevezetését a szolgáltatásaik közé.

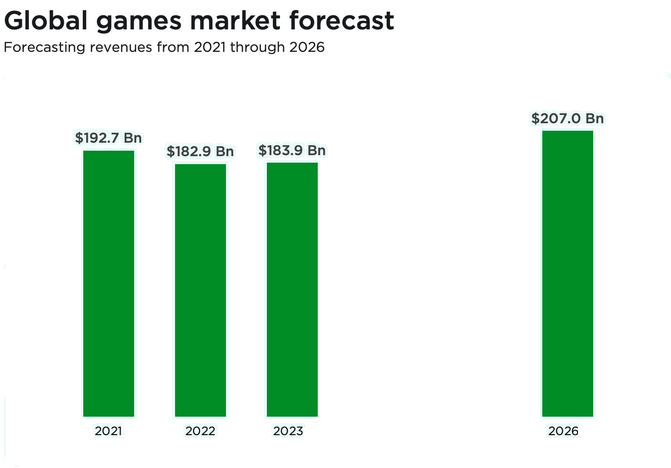
Fontosnak tartom még megemlíteni, hogy egyre nagyobb hangsúlyt kap a VR és AR technológia térhódítása is. A virtuális valóság (VR) és a kiterjesztett valóság (AR) technológiák számottevő hatást gyakorolnak a játékfejlesztésre, új dimenziókat nyitva a játékélmények terén. A VR headsetek, mint az Oculus Rift, a HTC Vive, és a PlayStation VR, lehetővé teszik a játékosok számára, hogy teljesen belemerüljenek a digitális világba, ahol szinte kézzelfoghatóan élhetik át a játékokat. Ezek a technológiák valós idejű, 360 fokos látványt és interaktív élményeket biztosítanak, amelyek messze túlmutatnak a hagyományos képernyős játékok kínálta lehetőségeken. *[3]*

Összefoglalva a játékfejlesztés hosszan tartó és változó történetén keresztül az emberiség mindig is arra törekedett, hogy izgalmas és figyelemfelkeltő játékokat hozzanak létre, amelyek segítenek a szórakozásban, az unalom elűzésében, és a kikapcsolódásban. A játékfejlesztés folyamatosan fejlődik és változik, és a jövőben is számos izgalmas lehetőséget fog kínálni mind a játékosok, mind a fejlesztők számára.

# 3. A játékpiac helyzete

## 3.1 Történelmi áttekintés

A videojátékok piaca az egyik leginkább növekvő szektor a szórakoztatóipar területén. Az elmúlt évtizedek alatt a videojátékok jelentős átalakuláson mentek keresztül, mind technológiai, mind gazdasági szempontból nézve. Ez a terület ma már nemcsak a gyerekek és a tinédzserek szórakozását szolgálja, hanem széles körű csoportokat vonz, beleértve a felnőtteket és az idősebb generációt is. Az egyszerű, kétdimenziós játékoktól kezdve a mai napig, amikor már fotórealisztikus grafikákat és összetett játékmechanikákat nyújtanak, a videojátékok rendkívül sokat alakultak, változtak.

A piacnak az értéke 2023-ben meghaladta a 183 milliárd dollárt, és az előrejelzések szerint ez az összeg tovább fog növekedni az elkövetkező években. 2026-ra 207 milliárd dollárra számítanak. Bizonyos bevétel-előrejelzések alapján pedig 583 milliárd dollárt becsülnek 2030-ra. *[6]*

1.ábra: Nezetközi játékpiac előrejelzése

Az elmúlt tíz évben a videojáték-ipar folyamatos növekedésen ment keresztül. 2013-ban a globális videojáték-piac értéke körülbelül 85 milliárd dollár volt és 2023-ra meghaladta a 183 milliárd dollárt, ami több mint kétszeres növekedést jelent. Ez a fejlődés a technológiai újításoknak, az új üzleti modelleknek és a növekvő felhasználói bázisnak köszönhető. Ha visszatekintünk 20 évvel korábbra: 2003-ban a piac értéke 52 milliárd dollár is tökéletesen kirajzolja a hosszú távú növekedést és a pénzügyi bevételek bővülését a játékiparban. *[5]*

## 3.2 A piac hajtóereje

**Mobil játékok térnyerése:** A mobil játékok az elmúlt években a videojáték-piac egyik legjelentősebb hajtóerejévé váltak. Az okostelefonok és táblagépek elterjedése lehetővé tette, hogy a játékok széles közönséghez jussanak el. A mobil játékok könnyen hozzáférhetők, gyakran ingyenesen letölthetők, és egyszerűen integrálhatók a mindennapi életbe, így egyre több ember számára jelentenek kikapcsolódási lehetőséget. Az olyan játékok, említésként mint a Candy Crush Saga, a Clash of Clans és a Pokémon GO hatalmas sikert arattak, és milliárd dolláros bevételeket generáltak. A mobil játékok pénzügyi modelljei, amelyeket gyakran mikro tranzakciókra és reklámokra építenek, tovább növelik a fejlesztők és kiadók jövedelmezőségét. A mobil játékok gyakran úgy vannak tervezve, hogy rövid ideig tartó, de gyakori játmenetet kínáljanak, ami passzol a felhasználók napi rutinjához. *[3]*

Tehát a mobil játékok közkedveltsége mögött álló tényezők:

* Hozzáférhetőség
* Ingyenes letöltések
* Rövid játékidő

**E-sportok berobbanása:** Az e-sport, vagyis a kompetitív videojátékok felkapottsága hihetetlenül gyarapodott az elmúlt években. Az e-sportversenyek hatalmas közönséget vonzanak, a legnagyobb események, több millió online és élő nézőt hívogatnak. Az e-sport piac bevételei elsősorban szponzorálásból, reklámozásból, jegyértékesítésből származnak. Fontos vonzerőt jelent a játékosok és csapatok hírneve, valamint a játékok pénzdíja is. Az e-sport ökoszisztéma kialakulása és fejlődése új bevételi forrásokat és marketing lehetőségeket teremtett a játékfejlesztők számára is. *[3]*

Felkapottságának szempontjai:

* Professzionális versenyzés
* Közösség
* Élő közvetítések
* Jókora reklámozási lehetőség

**Felhőalapú játékok:** A felhőalapú technológia felhasználásával is futtathatunk már játékokat bármilyen eszközön anélkül, hogy nagy teljesítményű hardverre lenne szükségünk. A Microsoft xCloud és az NVIDIA GeForce Now olyan szolgáltatásokat nyújtanak, amelyekkel a játékokat futtatni tudjuk távoli szervereken, és a tartalmat megosztani az interneten keresztül a játékosok eszközeire. Ez a technológia különösen csalogató azok számára, akik nem rendelkeznek nagy teljesítményű játékkonzollal vagy PC-vel, de továbbra is szeretnék élvezni a legújabb, nagyobb hardverigénnyel rendelkező játéktartalmakat. A felhőalapú játékok előnyei közé tartozik a hozzáférhetőség, a költséghatékonyság és a platformfüggetlenség, amely utat nyit a játékosok számára, hogy bárhol és bármilyen eszközön játszanak. *[3]*

Keresettségének okai:

* Könnyű hozzáférhetőség
* Költséghatékonyság
* Alacsony hardverigény

**Előfizetéses modellek:** Az előfizetéses modellek egyre nagyobb előretörést érnek el a videojáték-iparban. Az olyan szolgáltatások, mint a Microsoft Xbox Game Pass, a PlayStation Plus, a Ubisoft+ és az EA Play széles játékkönyvtárhoz biztosítanak hozzáférést rögzített havi díj ellenében. Ezen megoldások a játékosok számára költséghatékony hozzáférési lehetőséget adnak számos játékhoz, míg a fejlesztők és kiadók számára folyamatos bevételi forrást jelentenek és hosszú távú lojalitást szerezhetnek a felhasználók részéről. Ezek a modellek használatával a játékosok új játékokat próbálhatnak ki anélkül, hogy teljes árat kellene fizetniük érte, és így próbálják elérni a játékok népszerűsítésének és használatának elterjedését.

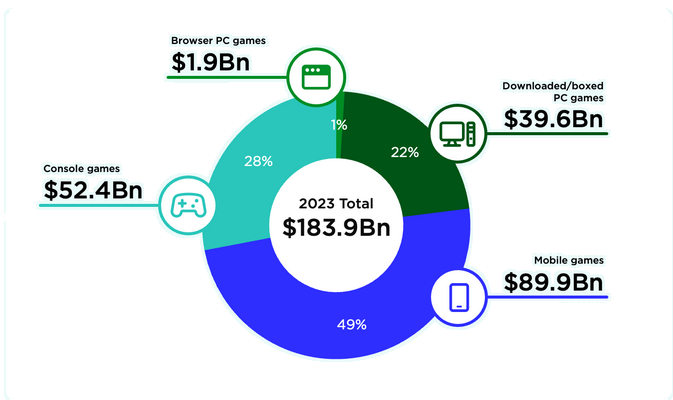
Popularitásának összetevői:

* Folyamatos frissítések, gyakori megújulás
* Költséghatékonyság
* Hűség a szolgáltatás felé
* Bevételbiztonság (fix havi bevétel)

Mindent egybevetve a piac növekedését számtalan tényező hajtja, köztük a mobil játékok térnyerése, az e-sportok népszerűsége, a felhőalapú játékok elterjedése és az előfizetéses modellek előretörése. Ezek nemcsak új lehetőségeket teremtenek a fejlesztők és kiadók számára, hanem a játékosok számára is gazdagabb és változatosabb élményeket nyújtanak, ezáltal tovább erősítve a videojáték-ipar pozícióját a szórakoztatóipari szektorban.

## 3.3 A piac eloszlása

A játékipar jelentős mértékben eltér az eszközök típusa alapján, amelyeken a játékokat játszák. Az alábbiakban bemutatom a legfontosabb eszközkategóriákat és azok piaci részesedését:



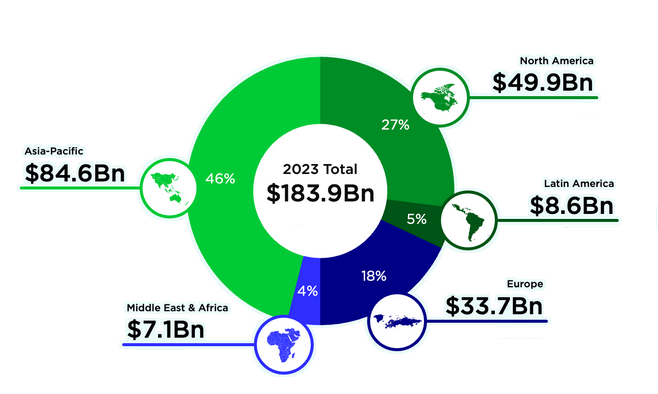
2.ábra: Globális játékpiaci bevétel 2023-ban eszközök szerint [6]

**Mobil Játékok**: A legnagyobb piaci egységet képezik. 2023-ban a mobil játékok a teljes piac közel felét: 49%-át tették ki. Ennek a szegmensnek a növekedését a széleskörű okostelefon-használat, a könnyen hozzáférhető alkalmazásboltok, és az ingyenesen elérhető (free-to-play) modellek népszerűsége indokolja. A piaci részesedése: 89,9 milliárd dollárt ért el. *[4] [6]*

**Konzolos Játékok**: Ez a kategória foglalja el a második helyezést a területen, különösen a prémium konzolok (PlayStation, Xbox, Nintendo Switch) esetében. Ezek az eszközök a teljes bevétel körülbelül 28%-át teszik ki. A konzolos játékok előnye, hogy magas színvonalú grafikai élményt nyújtanak, és gyakran kínálnak exkluzív (csak a platformon elérhető) címeket. A konzolok részesedése: 52,4 milliárd dollárt ért el 2023-ban. *[6]*

**PC Játékok**: A piaci részesedése a PC játékoknak az elmúlt években mindig 20-25% körül mozgott. A tavalyi kimutatás alapján a 23%-ot érte el, amibe beletartozik a böngészőből futó játékoktól kezdve a telepíthető pc játékokig minden. Az asztali számítógépeken és laptopokon játszott játékok különösen közismertek a keményvonalas játékosok körében, akik a nagy teljesítményű hardvereket és a széleskörű testre szabási lehetőségeket értékelik a leginkább. Az online elérhető értékesítési helyek jelentősen hozzájárultak ennek a területrésznek a fejlődéséhez. A PC játékok: 41,5 milliárd dollárt értek el a mért adatok alapján. *[6]*

A videojáték-piac eloszlása földrajzi szempontból is megfigyelésre érdemes:



3.ábra: Globális játékpiaci bevétel 2023-ban régiók szerint [6]

* **Megfigyelhető, hogy Ázsia, főképp Kína és Japán** a legnagyobb és legdinamikusabban gyümölcsöző videojáték-piacot képviselő ország. Kína önmagában a globális bevételek több mint egyharmadát generálja, főként a mobil játékok révén. Japán pedig a konzolos és mobil játékok egyik legnagyobb piaca. Így a bevételek 46%-át adják az iparnak.
* A következő nagy bevételt jelentő kontinens, nem más mint Észak-Amerika 27%-al. Ez a régió a konzolos és PC-s játékok értékesítése terén rendkívül erős.
* Európa szintén számottevő piaci részesedéssel bír (18%), különösképp az Egyesült Királyság, Németország, és Franciaország révén. Az európai piac sokszínűsége miatt a konzolos, PC-s, és mobil játékok egyaránt népszerűek.
* Bár latin-amerikai, a közel-keleti és afrikai régió kisebb nyereséget jelentenek a játékipar számára, azonban a növekedési potenciáljuk hatalmas. A mobil játékok különösen népszerűek, mivel az okostelefonok terjednek leginkább ebben a térségben megtalálható országokban. *[3] [6]*

# 4. A játékfejlesztésnél használt alapfogalmak

A videojátékok fejlesztése egy komplex folyamat, amely számos technológiai eszközt és szoftvert igényel. A játékmotorok, fejlesztési keretrendszerek és egyéb technológiák együttesen járulnak hozzá a hatékony játékfejlesztéshez. A következőkben ezek közül nagy hangsúlyt kap a játékmotor, amely központi szerepet játszik a játékélmény javításában.

**Játékfejlesztési motor (Game Engine):**

A játékmotorok olyan speciális szoftverplatformok, amelyek segítségével a játékok különféle technikai aspektusainak fejleszthetjük, például a grafika, a fizika, a hangok és az animáció kezelését. Az alapvető grafikai modul felelős a 2D és 3D objektumok rendereléséért, amelyek megjelenítik a játék világát, karaktereit és eseményeit. A fizikai motor a valóság törvényeit szimulálja. Például gravitáció, ütközések és a mozgás viselkedése, így a játékélmény még valósághűbbé válik. A hangmotor a különböző hangeffektek és zenék irányításáért felelős, amelyek fokozzák a játékélményt. A legtöbb játékmotor tartalmaz animációs és mesterséges intelligencia (AI) modulokat is. Az animációs rendszer a karakterek és objektumok mozgásának zökkenőmentes kivitelezéséért felel, míg az AI lehetőséget biztosít arra, hogy az ellenségek és NPC-k intelligensen reagáljanak a játékos lépéseire. Ezen felül a scripting lehetőségek révén a fejlesztők könnyedén megírhatják a játékmenet logikáját, így egyedi interakciókat és eseményeket valósíthatnak meg. A legismertebb játékmotorok közé tartozik az Unity, az Unreal Engine, a CryEngine és a Godot. A motorok olyan átfogó eszközkészletet biztosítanak, amelyek minden területet lefednek, ami a professzionális játékfejlesztéshez szükségesek. Ezek a motorok gyakran több platformot is támogatnak, így a játék egyszerűbben portolható különböző eszközökre, ami fontos előnyt jelent a fejlesztés során. *[7] [8]*

**Fejlesztési keretrendszer (Development Framework):**

Olyan általános célú eszközkészletek, amelyek a szoftverfejlesztés több területén is alkalmazhatók, beleértve a játékfejlesztést is. Ezek a keretrendszerek olyan alapvető funkciókat és kódkönyvtárakat biztosítanak, amelyek használatával a fejlesztők az ismétlődő feladatokat gyorsan és hatékonyan tudják megoldani. *[7]*

Tehát elmondható, hogy a játékfejlesztési motorok nélkülözhetetlen eszközök a modern videojáték-fejlesztésben. A különféle modulok révén a fejlesztők minden szükséges technológiához hozzáférnek, amely egy professzionális játék létrehozásához szükséges, a grafikai megjelenítéstől a hangkezelésen át a mesterséges intelligenciáig. A játékmotorok sokoldalúsága és a keretrendszerek modularitása gyanánt a fejlesztők minden szempontból optimalizált és felhasználóbarát játékot készítsenek.

# 5. Játékfejlesztési motorok

A fejlesztésben rengeteg technológia áll rendelkezésre, amelyek által a fejlesztők különböző platformokra és célközönséghez szóló játékokat hozhatnak létre. Ezek közül bemutatom, jellemzem a legnépszerűbb játékfejlesztési motorokat:

**Unity:**

* Az egyik legszélesebb körben használt játékfejlesztő motor, amely támogatja a PC-t, játékkonzolokat, mobileszközöket és VR/AR platformokat. Segít több platform egyidejű fejlesztésében is. A Unity több mint 25 platformot támogat.
* Egyszerűen elsajátítható, különösen a kezdők számára. A felhasználói felület intuitív, és rengeteg oktatási anyag áll rendelkezésre. Tanulhatóságának köszönhetően rövid időn belül hatékony játékokat készíthetünk vele.
* A fejlesztők programkódokat írhatnak a C# programozási nyelv használatával, amely egy viszonylag könnyen megtanulható.
* A Unity Asset Store lehetővé teszi a fejlesztők számára, hogy előre elkészített 3D modelleket, textúrákat, animációkat és egyéb eszközöket vásároljanak, ami jelentősen felgyorsíthatja a fejlesztés menetét.
* Nem csak játékokhoz, hanem interaktív 3D alkalmazásokhoz, szimulációkhoz és VR/AR projektekhez is ideális megoldást kínál.
* Népszerű játékok, amiket Unityben készítettek: Hollow Knight, Pokemon GO, Angry Birds 2. *[9] [10]*

**Unreal Engine:**

* Az Unreal Engine kiemelkedő grafikai minőséget nyújt fejlett renderelési képességeinek köszönhetően, így az AAA játékok fejlesztésében népszerű választás.
* Egyedi Blueprints rendszerével lehetőséget ad arra, hogy a fejlesztők intuitív módon, programozási ismeretek nélkül is megteremthessék a játékmenetet és a logikai rendszereket.
* A beépített Unreal Physics motor valósághű fizikát és interakciókat valósít meg, ezáltal fokozva a játékélményt.
* A C++ programozási nyelvet alkalmazza, amely magas teljesítményt és nagyfokú rugalmasságot nyújt, bár összetettebb, mint a Unity által használt C#.
* Nyílt forráskódú megoldásként a fejlesztők számára jelentős testreszabási lehetőségeket kínál.
* Kiterjedt eszközkészlete támogatja a fejlesztés minden szakaszát, és könnyen összekapcsolható más rendszerekkel.
* Az Unreal Engine nem csupán játékok, hanem különféle alkalmazások létrehozására szintén alkalmas, így széles körben alkalmazható.
* Kiválóan támogatja a VR- és AR-fejlesztést is, gazdag integrációs és optimalizálási lehetőségeivel, amelyek célja a kiváló felhasználói élmény megteremtése.
* Fotorealisztikus vizualizációs képességei miatt nemcsak játékok, hanem filmek és egyéb médiaipari alkalmazások esetén is nagyszerűen használható. *[9] [11]*



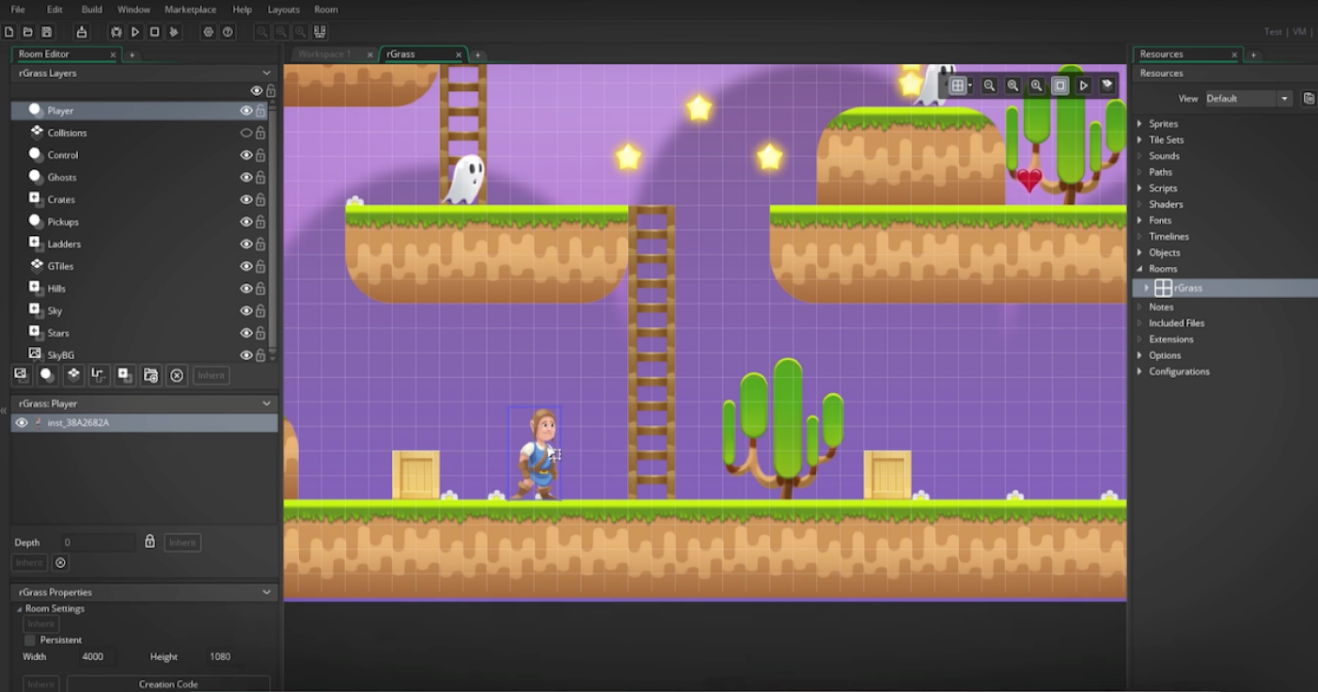
4.ábra: Unreal Engine játéktervezői felülete

**Godot Engine:**

* Nyílt forráskódú és ingyenes: A Godot Engine egy teljesen ingyenes, nyílt forráskódú szoftver, amely lehetővé teszi a fejlesztők számára, hogy szabadon módosítsák és saját igényeikhez igazítsák az eszközt. A nyílt forráskód miatt a felhasználói közösség folyamatosan fejleszti és karbantartja.
* Támogatja a 2D és 3D játékkészítést: Kiválóan alkalmas 2D és 3D játékok készítésére. Különösen erős 2D-támogatással rendelkezik, így ideális platformerek, kirakós játékok és egyéb 2D játékok készítéséhez.
* A GDScript nyelvet használ: A saját GDScript programozási nyelvét használja, amely hasonló a Pythonhoz. A nyelv könnyen megtanulható és használható, és támogatja az objektumorientált, imperatív és funkcionális programozási paradigma alkalmazását.
* Széleskörű többplatformos támogatásával garantálja játékok létrehozását és exportálását több platformra, beleértve a Windowst, a macOS-t, a Linuxot, az Androidot, az iOS-t és a HTML5-öt.
* A motor felhasználásával készült játékok: Heartbeat, Golf Peaks. *[9]*

**GameMaker Studio:**

* Célközönség: A GameMaker Studio elsősorban kezdő és középhaladó szintű játékfejlesztőknek készült, széleskörű programozási ismeretek nélkül.
* Az eszköz használata könnyen megtanulható, így a felhasználók viszonylag gyorsan hozhatnak létre játékokat anélkül, hogy nagy programozási szakértelemre lenne szükségük.
* Drag-and-drop interface-el rendelkezik. Az alkalmazás drag-and-drop felületet kínál, amely lehetővé teszi a felhasználók számára, hogy viszonylag egyszerűen alakítsanak ki játékmenetet és logikát mély kódolási ismeretek nélkül.
* Platformfüggetlenségének köszönhetően a játékfejlesztők több rendszerre exportálhatják játékaikat, ideértve a PC-ket, konzolokat és mobileszközöket is.
* Az itt készült játékok: Undertale, Hotline Miami. *[12]*

****

5.ábra: GameMaker Studio játékfejlesztői felülete

**CryEngine:**

* A grafikai teljesítmény és a valósághű megjelenítés terén az egyik legjobb választás lehet. A legújabb verziók összetett fényhatásokat, részletes textúrákat és valósághű animációkat tartalmaznak.
* Egy sor fejlett funkciót és eszközt kínál a fejlesztőknek, beleszámolva a valós idejű fizikai alapú szimulációt, az AI-eszközöket és a nagyobb méretű világok támogatását.
* Hátránya, hogy általánosságban összetettebb és nehezebben használható, mint a például a GameMaker Studio, és a tapasztaltabb fejlesztőknek készült.
* C++ és C# programozási nyelveket használ.
* Ideális választás lehet nagyobb költségvetésű projektekhez, ahol fontos a grafikai minőség és a méretezhetőség.
* CryEngine használatával alkotott videójátékok: Far Cry, Crysis, Star Citizen. *[9]*

# 6. HTML5 játékfejlesztési technológiák

A HTML5 játékfejlesztés az elmúlt években egyre népszerűbbé vált, mivel lehetőséget kínál, hogy játékokat fejlesszenek webalapú platformokra, amelyek széles körben elérhetők asztali és mobileszközökre egyaránt. A technológia, amely a játékok szerkezetét és megjelenését biztosítja, maga a HTML5. Ez beépített elemeket kínál, mint például a <canvas> elem, amely hozzájárul a grafikai tartalom rajzolásához közvetlenül a böngészőben. A HTML5 játékok alapvető programozási nyelve a JavaScript, ezzel kezelhetők az interakciók, az animációk, a játék logikája és a felhasználói inputok. A modern JavaScript motorok, mint például a Google Chrome V8 motorja, jelentős teljesítménynövekedést eredményeztek a böngésző alapú játékokban, ami kedvez a zökkenőmentes játékélmény kialakításához.

A HTML5 egyik legfontosabb eleme a fejlesztés szempontjából a Canvas API. A <canvas> elem biztosítja 2D grafikák renderelését (grafikus megjelenítését) JavaScript segítségével, ami az alapját képezi a rajzolási műveleteknek, például a sprite-ok megjelenítésének, az animációknak és egyéb vizuális elemeknek. Ha 3D grafikáról van szó, a WebGL (Web Graphics Library) lehetőséget nyújt arra, hogy JavaScript API segítségével 3D grafikákat jelenítsenek meg a böngészőkben, anélkül, hogy bővítményeket kellene telepítenünk. A WebGL emellett lehetővé teszi a GPU kihasználását, amit a különösen látványos és teljesítményigényes játékok esetében ezt előnyös kihasználni. Az ilyen típusú játékokban a hangok és zenék integrálása elengedhetetlen, és ebben nyújt segítséget a HTML5 Audio API. Ezen API-k révén a fejlesztők könnyedén kezelhetik a hanghatásokat, háttérzenéket és egyéb hangalapú interakciókat, amelyek az élmény fokozásához járulnak hozzá.

Számos keretrendszer és könyvtár áll rendelkezésre, amelyek megkönnyíthetik és felgyorsíthatják a fejlesztés folyamatát, különösen azok számára, akiknek fontos a hatékonyság és a rugalmasság. Ezek a keretrendszerek és könyvtárak rengeteg előre elkészített funkciót kínálnak, amelyekkel a fejlesztők gyorsabban hozhatnak létre játékokat, kevesebb kódolási munka mellett.

## 6.1 Ismertebb keretrendszerek és könyvtárak

* **Phaser.js** az egyik legismertebb és legszélesebb körben használt HTML5 játékmotor, amely kifejezetten 2D játékok fejlesztésére lett kitalálva. A Phaser kiváló választás azoknak, akik gyorsan szeretnének látványos és jól működő játékokat létrehozni, mivel rengeteg dokumentációval és egy aktív közösséggel rendelkezik, ami megkönnyíti az elakadt fejlesztők számára a segítségkérést és a problémamegoldást. A motor a beépített funkciók széles skáláját kínálja, a fizika szimuláción és animációkon keresztül, egészen a hangok kezeléséig. Így a fejlesztőknek nem szükséges ezeket az alapvető funkciókat saját maguknak implementálniuk. *[13] [14]*
* **Three.js** egy másik népszerű JavaScript könyvtár, amely a WebGL technológiára épül. Kifejezetten 3D grafikák egyszerű létrehozására lett tervezve, így nagyszerű választás 3D játékok fejlesztéséhez. A Three.js-el a fejlesztők bonyolult 3D-s jeleneteket hozhatnak létre, mint például dinamikus fényforrásokat, árnyékokat, textúrákat és animációkat. Ez a könyvtár nagy mértékben rugalmas, és számos előre elkészített példát, valamint bővítményt tartalmaz, amik segítenek abban, hogy a készítők gyorsabban és hatékonyabban valósíthassák meg a játékötleteiket. *[13] [14]*
* A **PixiJS** egy 2D renderelő könyvtár. Módfelett gyors rajzolási lehetőségeket kínál, így remek választás 2D játékok és interaktív alkalmazások készítéséhez. A PixiJS egyik fő előnye a teljesítményoptimalizálás, mivel az összetett grafikus elemek gyors megjelenítésére is alkalmas a böngészőkben. Ezt a teljesítményt a WebGL és a HTML5 Canvas kombinálásával éri el, és automatikusan kiválasztja a legmegfelelőbb technológiát a felhasználó eszköze alapján. Ez roppant hasznos lehet azoknak a fejlesztőknek, akik igényes grafikát szeretnének létrehozni anélkül, hogy a teljesítményben kompromisszumot kellene kötniük. *[13]*
* A **PlayCanvas** szintén WebGL technológián alapuló 3D-s játékmotor, amelyet szintén a böngészőben való futtatásra terveztek. A PlayCanvast az különbözteti meg a többitől, hogy van egy online szerkesztője, amely segíti a fejlesztőket abban, hogy valós időben együttműködjenek és megosztják munkájukat. Ez a motor effektív eszközkészletet ad, hozzákalkulálva a fizikai szimulációkat, a grafikai megjelenítést és az animációkat, így jó választás összetett 3D-s játékok készítéséhez. A másik előnye, hogy könnyen integrálható más webes technológiákkal, így rugalmasságot biztosít a játékok létrehozásához és közzétételéhez különböző platformokon. *[13] [14]*

Egybe véve, ezek a keretrendszerek és könyvtárak különböző típusú játékfejlesztési igényeket fednek le, legyen szó egyszerű 2D-s, vagy bonyolult 3D-s játékokról. Mivel minden megoldásnak megvannak a saját egyedi előnyei és hátrányai, így a fejlesztőknek célszerű a projektük igényeihez igazodva választaniuk a rendelkezésre álló eszközök közül.

## 6.2 HTML grafika: CANVAS, SVG

A modern webfejlesztés területén a **HTML5** számos grafikai elem megjelenítésre ad lehetőséget, két kiemelkedő technológiával: a **Canvas** és az **SVG** (Scalable Vector Graphics) alkalmazásával. Ezek a technológiák különböző módon kezelik a grafikák megjelenítését és interakcióját, így mindkettőnek megvannak az erősségei és gyengeségei, valamint a specifikus használati esetei.

A **Canvas** egy HTML5 elem, amely használatával a fejlesztők dinamikus, raszteres grafikákat hozhatnak létre JavaScript programozási nyelven. Ez a grafikai elem egy „rajzvásznat” biztosít, ahol különböző grafikai elemek – például vonalak, alakzatok, szövegek, képek és animációk – hozhatók létre. A canvas egyik erőssége, hogy teljes mértékben képpont alapú, vagyis minden egyes objektum egy adott képpontra rajzolódik ki, hasonlóan a képszerkesztő programok működéséhez. *[15]*

A technológia legfőbb előnye a nagyfokú rugalmasság és a gyorsaság, amely különösen hasznos, amikor valós idejű grafikai frissítésekre, például játékokban animációkra vagy adatvizualizációra van szükség. Mivel a canvas képekkel és képpontokkal dolgozik, a grafikák egyszerűen, közvetlenül manipulálhatók JavaScript kódon keresztül, így komplex és interaktív grafikai megoldások alkothatók meg. Ezen túlmenően támogatja az olyan további funkciókat, mint a pixelmanipuláció, amellyel tudjuk módosítani az egyes pixelek színeit, átlátszóságát vagy más tulajdonságait. Ez jól jöhet képek szűréséhez vagy különleges effektusok létrehozásához. Egy másik kulcsfontosságú előnye a valós idejű animáció létrehozásának lehetősége, amely különösen a dinamikus, mozgó tartalmak megjelenítésekor fontos. Mivel a képernyő közvetlenül frissíthető, a canvas lehetőséget kínál arra is, hogy gyors és folyékony animációkat hozzunk létre anélkül, hogy a teljes rajz újra generálására szükségünk lenne. *[15]*

Azonban a képpont alapú megközelítés negatívuma, hogy ha a vásznon megrajzolt grafikát nagyítjuk vagy kicsinyítjük, az veszít a minőségéből, mivel nem méretezhető át tökéletesen, ahogy a vektoros grafikák. Továbbá, a canvas statikus természete miatt nem tartja nyilván az egyes rajzelemeket, vagyis ha egy elemet újra szeretnénk rajzolni vagy frissíteni, akkor a teljes vásznat újra kell rajzolni. Ez nagyobb adatmennyiségnél vagy összetett grafikai elemeknél teljesítménybeli problémákhoz vezethet. *[15]*

Ezzel szemben a **Scalable Vector Graphics (SVG)** egy másik HTML5 megoldás a grafikák megjelenítésére, de a canvas-szal ellentétben az SVG vektoralapú. Ez azt jelenti, hogy az SVG-vel megjelenített grafikus elemek matematikai képletek, nem pedig pixelek alapján jönnek létre. Tehát a grafika nagyítás vagy kicsinyítés könnyűszerrel megoldható bármilyen méretre a minőség romlása nélkül. Az SVG előre definiált elemeket, úgy mint vonalakat, köröket, négyzeteket, sőt, XML formátumban tárolt összetett formákat is képes megjeleníteni. Előnyei fokozottan szembetűnőek a statikus, jó minőségű grafikai elemek, például logók, diagramok, ikonok vagy más vektoros alakzatok kreálásakor. A vektor alapú megközelítéssel egyszerű az elemek automatikus átméretezése ahhoz, hogy illeszkedjenek a böngészőablak vagy más képernyő méretéhez anélkül, hogy elveszítenék a részleteket. Ezen kívül könnyen módosítható CSS-sel és JavaScripttel egyaránt, így az interaktív grafikák kialakításában is szerepet játszhat. *[15] [16]*

Canvas és SVG összehasonlítása:

A **canvas** és az **SVG** közötti legnagyobb különbség az alapvető megközelítésben rejlik: míg a canvas képpont alapú (raszteres), addig az SVG vektoros formátumot használ. Emiatt a canvas jobban használható valós idejű, folyamatosan változó grafikák, például játékok, animációk vagy diagramok megjelenítéséhez, ahol a grafikai elemek gyorsan és folyamatosan kell, hogy frissüljenek. Az SVG pedig inkább statikus vagy ritkábban frissülő grafikákhoz alkalmas, ahol a részletesség és a minőség fontosabb, például ikonok, logók vagy interaktív diagramok esetén. *[15]*

**Canvas** előnyei közé tartozik a nagyfokú rugalmasság és a valós idejű grafikai frissítések, azonban hátránya a minőség romlása nagyításkor és a képpontok korlátozott kezelhetősége.

**SVG** fölénye, hogy a grafikai elemek bármilyen méretben részletgazdagok maradnak, és könnyen integrálhatók a HTML dokumentum szerkezetébe. Hátránya viszont, hogy nagyobb és összetettebb grafikák esetén teljesítményproblémák léphetnek fel, főképp, ha sok elemről van szó, vagy gyakori frissítést igényel. *[15]*

Felhasználási esetek:

A canvas használata ideális videojátékok fejlesztésénél, ahol szükség van a gyors, dinamikus grafikai frissítésekre, mint amilyenek a mozgó karakterek vagy háttéranimációk. Továbbá, valós idejű animációkban, mint például interaktív vizuális effektek, vagy adatvizualizációk során is hasznos, ahol a teljesítmény kulcsfontosságú. Ezzel ellentétben az SVG leginkább akkor alkalmazható, amikor statikus grafikákat kell létrehozni, amik megőrzik éles vonalaikat és részleteiket minden felbontásban. Feltételezhetően megfelelő ikonok, logók, vagy diagramok készítésénél, ahol a méretezhetőség és a minőség megőrzése elengedhetetlen. *[15]*

## 6.3 Modern webes adattárolási megoldások

A témában említésre méltó még a **Local Storage, SessionStorage és az IndexedDB** szerepe amelyek, olyan esszenciális HTML5 eszközök, amelyek utat engednek a fejlesztő számára, hogy a webalkalmazásokban helyi adattárolást valósítsanak meg a felhasználók böngészőjében. Ezek a technológiák nagy előrelépést jelentenek a felhasználói behatások erősítésében, különösen akkor, amikor internetkapcsolat nélküli működésre, vagy gyors, kiszolgálófüggetlen adattárolásra van szükség. *[17]*

A **Local Storage** az egyik legkezénfekvőbb megoldás a kliensoldali adattárolásra. A HTML5 bevezetésével ez a technológia egy tartósabb megoldást nyújtott a régebbi cookie-alapú tárolás helyett, mivel nagyobb adatmennyiséget (5-10 MB böngészőtől függően) képes kezelni, és nem kerül minden egyes szerverrel való kommunikáció során elküldésre, mint a cookie-k. A Local Storage egy egyszerű kulcs-érték páros rendszert alkalmaz, ahol az adatok a használt böngészőben maradnak akkor is, ha a felhasználó bezárja az alkalmazást vagy újraindítja a böngészőt. *[17]*

A Local Storage kiváltképpen alkalmas olyan kisebb adatok tárolására, mint például:

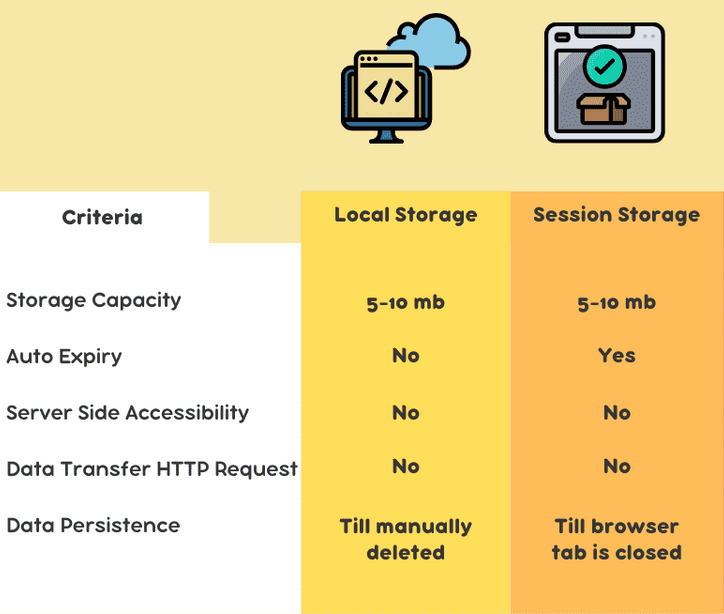
* Felhasználói beállítások (pl. témaválasztás, nyelvi preferenciák)
* Egyszerű statisztikai adatok (pl. elért pontszámok játékban)
* Munkamenet előzményei vagy cache adatok

Legfőbb hátránya, hogy az adatok csak szöveges formátumban tárolhatók, így komplex objektumokat JSON formátumba kell konvertálni tárolás előtt, majd újra vissza kell alakítani, ha használni szeretnénk. *[17]*

A **Session Storage** a HTML5 másik kliensoldali adattárolási megoldás, amivel az adatok ideiglenes tárolhatók egy adott böngészési munkamenet idejére. Ez az alternatíva hasonló a Local Storage-hoz, de alapvető különbség, hogy az itt tárolt adatok csak addig maradnak meg, amíg a felhasználó nem zárja be a böngészőablakot. Miután a munkamenet lezárult, az összes adat automatikusan törlődik, így garantálva, hogy az ideiglenes információk ne maradjanak feleslegesen letárolva. Ezt a tulajdonságot számos felhasználási esetben kihasználhatjuk, amikor átmeneti adattárolásra van szükség.

A Session Storage tipikusan olyan helyzetekben használható, ahol az adatokra csak egy adott böngészési munkamenet során van szükség. Példaként egy weboldalon a felhasználó bejelentkezési állapotát követhetjük egy böngészési fülön belül anélkül, hogy az adatok megmaradnának a fül bezárása után. Emellett gyakran használják több lépésből álló űrlapoknál, ahol az adatoknak meg kell maradniuk az oldalak közötti navigáció során, de már nem szükségesek, miután a folyamat befejeződik. Azonfelül alkalmas még átmeneti „bevásárlókosár” adatok tárolására, és minden olyan dinamikus tartalomnál, ahol csak egy adott munkamenet alatt van rá szükségünk. *[18]*

Erőssége, hogy az adatok automatikusan törlődnek a munkamenet végén, így nem kell aggódni a régi vagy felesleges információk eltávolítása miatt. Ezzel egy szimpla és kényelmes megoldást kínál a rövid távú adatkezelésre, mivel az adatokat nem kell kézzel törölni vagy frissíteni, amikor azok már nem szükségesek. Ezen felül biztosítva van, hogy az adatok különállóan kezelhetők legyenek a különböző böngészőfülek között, azaz az egyik fülben tárolt adatok nem lesznek elérhetők egy másikban. Azonban ez is a hátránya, mivel az adatok csak egy munkamenet alatt érhetők el, nem alkalmas hosszú távú adatmegőrzésre. Így például felhasználói beállítások, játékállások vagy egyéb, tartós adattárolást igénylő információk mentésére a Session Storage nem megfelelő. Emellett, bár az adatmennyisége általában szintúgy 5-10 MB között van, amely elegendő lehet kisebb adatokhoz, nagyobb vagy összetettebb adatok kezelésére nem a legjobb megoldás, különösen olyan esetekben, ahol strukturált vagy relációs adatok tárolására van szükség. *[18]*



6.ábra: Local Storage és a Session Storage összehasonlítása [18]

**Az IndexedDB** egy fejlettebb böngésző alapú adatbázis, amely nagy mennyiségű adat tárolására és kezelésére alkalmas. Ez egy NoSQL-alapú adatbázis, amely aszinkron módon működik, és támogatja strukturált adatok, például objektumok, fájlok és bináris adatok tárolását. *[19]*

Az IndexedDB különösen alkalmas összetett feladatokhoz:

* Nagy mennyiségű adatot tárol, így játékhelyekhez, adatbázisokhoz, képekhez, videókhoz vagy gyorsítótárazott-fájlokhoz.
* Lokálisan futó alkalmazásokhoz (offline módban is használható)
* Alkalmazásokhoz, amelyek megkövetelik, hogy az adatokat strukturált és relációs szerkezetben tárolják

Az IndexedDB tranzakciók, lekérdezések és indexek alkalmazását támogatja, így az adatok keresése és módosítása hatékonyabbá válik. Bár a használata összetett, rugalmassága révén a fejlesztők számára jelentős kontrollt biztosít. *[19]*

**Local Storage és IndexedDB összehasonlítása:**

**Adatmennyiség szempontjából** a Local Storage korlátozottan, néhány megabájt adat tárolására képes, míg az IndexedDB gyakorlatilag korlátlan mennyiségű adatot képes kezelni (a böngésző által megengedett határok között).

**Használat egyszerűségének perspektívájából** a Local Storage könnyebben használható, mivel egyszerű kulcs-érték párokat használ, míg az IndexedDB komplexebb API-t igényel.

**Teljesítmény szemszögéből** pedig az IndexedDB nagyobb adatmennyiségek és összetett adatszerkezetek kezelésére optimalizált, míg a Local Storage egyszerűbb feladatokra alkalmas.

Mindhárom technológia kimagasló szerepet játszik a modern webalkalmazásokban, ahol fontos a hatékony helyi adattárolás. Az egyszerűbb, tartós adatokhoz, mint a felhasználói beállítások vagy a játékban elért pontszámok, a **Local Storage** teljes mértékben megfelelő. Ha viszont ideiglenes adatokat szeretnénk tárolni, például egy munkamenet alatt használt adatokat, amelyek a böngésző bezárásával azonnal törlődnek, a **Session Storage** kínál praktikus megközelítést. A komplexebb, strukturált adatokhoz, példaként egy játékállás mentéséhez vagy egy offline alkalmazás működtetéséhez, az **IndexedDB** biztosít bővíthető, rugalmas és hatékony adattárolási lehetőséget. *[17] [18] [19]*

**A WebSockets technológia:**

Alapvető szerepet játszik a HTML5 játékfejlesztésben: főleg az online multiplayer játékoknál. A WebSocket egy olyan protokoll, amely létrehozza a valós idejű, kétirányú kommunikációt a kliens és a szerver között. Ez azt jelenti, hogy a játékosok közötti interakciók azonnal megjelennek a játékban, függetlenül attól, hogy hol helyezkednek el a világban. Ezen felül a WebSockets hatékonyabb adatátvitelt biztosít, mint a hagyományos HTTP-alapú kommunikáció, mivel a kapcsolat egyszer létrejön, majd nyitva marad, így az üzenetek küldése és fogadása gyorsabban és kevesebb késéssel történik. Ez alapvető fontosságú a sima és zökkenőmentes játékélmény biztosításához, különösen azokban a játékokban, ahol minden egyes pillanat számít, például valós idejű stratégiai vagy lövöldözős játékokban. *[20]*

Összességében elmondható, hogy a HTML5 játékfejlesztés egyik legnagyobb előnye a platformfüggetlenség. A HTML5 játékok szinte minden modern böngészőben futtathatók, legyen szó asztali számítógépekről vagy mobil eszközökről. Emellett a fejlesztéshez szükséges eszközök és technológiák átfogó mennyiségben hozzáférhetők, és gyakran ingyenesek, ami különösen vonzóvá teszi őket a fejlesztők és kisebb stúdiók számára. Ez a kombináció lehetőséget teremt a játékfejlesztők számára, hogy nagy létszámú közönséghez juttassák el alkotásaikat, miközben minimálisak a fejlesztési költségeik és az időbeli ráfordításuk.

# 7. Digitális Tartalmak Felhasználása: Licencjogok és Fair Use

A digitális tartalom felhasználása és készítése során fontos tisztában lenni a tartalom felhasználásának szabályaival, különösképpen a tömérdek licencjogokkal, amelyek megszabják, hogyan és milyen feltételek mellett használhatók fel a képek, zenék és más digitális művészeti alkotások. Az alábbiakban bemutatok néhány a játékfejlesztésben használatos licencformát, és kifejtem azok jelentését és alkalmazását egyaránt.

## 7.1 Általános áttekintés

**CC (Creative Commons):**

Ezen licencek célja, hogy rugalmas opciókat biztosítsanak a szerzői jogok kezelésére, megengedve a művek széleskörű felhasználását, átalakítását és terjesztését. A CC licencek több típusra oszlanak, amelyek különböző feltételeket tartalmaznak a művek felhasználására vonatkozóan. Az alapvető követelmény, hogy a szerző nevét és a forrást megfelelően feltüntessék, de a specifikus licencek feltételei változhatnak attól függően, hogy milyen mértékű engedékenységet vagy korlátozást biztosítanak a felhasználók számára. A legújabb verziók, mint a CC 4.0, általában fejlettebb nemzetközi alkalmazhatóságot és korszerűbb szabályozást kínálnak az előző verziókhoz képest. *[21] [22]*

**OpenGameArt Attribution (OGA-BY 3.0 és OGA-BY 4.0)**

Az OGA-BY licenc a CC-BY licencből származik és kihagyja a CC-BY licencbe foglalt nehezen betartható kitételeket, hogy a művek szabadon felhasználhatók, módosíthatók és terjeszthetők legyenek, abban az esetben, hogyha az eredeti alkotó megfelelő kreditálása megtörténik. Ez a mű címéből és az alkotó nevéből áll. A kereskedelmi felhasználás is megengedett, de az esetleges módosításokat jelezni kell. A szerző kreditálása kötelező! *[22]*

**GPL (General Public License):**

Szabad szoftverlicenc, amely hozzájárul a szoftver szabad felhasználásához, módosításához és terjesztéséhez, feltéve, ha az eredeti licenc feltételeit megőrizzük. A GPL 3.0-s verziója fejlettebb jogi védelmet és nagyobb biztonságot nyújt a felhasználók számára a GPL 2.0-s verziójához képest, biztosítva a szabad szoftverek védelmét és terjesztését. A GPL licenc alatt megjelent művek használata általában nem engedélyezett kereskedelmi célú vagy zárt forrású projekteken, így a használójának alaposan utána kell járni a licenc feltételeinek mielőtt a művet használná. *[22]*

Ezek a licencek különböző jogosultságokat és korlátozásokat tartalmaznak, ezért fontos, hogy a projekt igényeinek megfelelő licenceket használjunk.

## 7.2 Creative Commons Licenc típusok

A Creative Commons (CC) licencek 6 különböző típusa közül választhatunk, az engedélyezési feltételek fokozatos szigorodása mellett. Az alábbiakban látható a különböző licencek jellemzői és feltételei: *[21]*

**CC0 (Creative Commons Zero)**

A CC0 egy közkinccsé tételi eszköz, amely használatával a szerzők lemondanak minden szerzői jogukról, és a művet a világszerte elérhető közkinccsé teszik. A CC0 lehetővé teszi a mű szabad felhasználását, módosítását és adaptálását bármilyen célra és bármilyen formában, feltételek nélkül. *[21]*

**CC BY (Creative Commons Attribution)**

Ezt a licencet a felhasználók szabadon terjeszhetik, módosíthatják, adaptálják és beépíthetik az alkotásukba bármilyen formában, feltéve, hogy a szerző nevét megfelelően megemlítik. Kereskedelmi célú felhasználás is megengedett. A CC BY licenc az alábbiakat tartalmazza:

* **BY:** A művet felhasználó személy vagy szervezet köteles feltüntetni a szerző nevét és az eredeti mű forrását. *[21]*

**CC BY-SA (Creative Commons Attribution-ShareAlike)**

Ez a licenc szabadon engedi a művek felhasználását, módosítását és adaptálását, feltéve, hogy a szerző nevét feltüntetik. A kereskedelmi felhasználás is engedélyezett. Amennyiben a művet átdolgozzuk, az új változatot azonos feltételek mellett kell licencelni. A CC BY-SA licenc a következő jogokat és kötelezettségeket tartalmazza:

* **BY:** A szerző nevét és a forrást meg kell említeni.
* **SA:** Az adaptált műveket ugyanazon feltételek mellett kell közzétenni. *[21]*

**CC BY-NC (Creative Commons Attribution-NonCommercial)**

Ez a licenc lehetővé teszi a művek szabad terjesztését, módosítását és adaptálását, azonban nem használható kereskedelmi célokra. A szerző nevét itt is meg kell említeni. A CC BY-NC licenc az alábbiakat tartalmazza:

* **BY:** A szerző nevét és a forrást meg kell említeni.
* **NC:** Csak nem kereskedelmi célú felhasználás engedélyezett. (Non-Commercial) *[21]*

**CC BY-NC-SA (Creative Commons Attribution-NonCommercial-ShareAlike)**

Ez a licenc hasonló a CC BY-NC licenchez, de további feltétel, hogy a módosított műveket azonos feltételek mellett kell közzétenni. A szerző nevét meg kell említeni, és a kereskedelmi felhasználás nem megengedett. A CC BY-NC-SA licenc az alábbiakat tartalmazza:

* **BY:** A szerző nevét és a forrást meg kell említeni.
* **NC:** Csak nem kereskedelmi célú felhasználás engedélyezett. (Non-Commercial)
* **SA:** Az adaptált műveket ugyanazon feltételek mellett kell közzétenni. *[21]*

**CC BY-ND (Creative Commons Attribution-NoDerivatives)**

Ez a licenc engedélyezi a mű másolását és terjesztését bármilyen formában, de csak az eredeti, változatlan formában. A kereskedelmi felhasználás engedélyezett. A szerző nevét meg kell említeni. A CC BY-ND licenc az alábbiakat tartalmazza:

* **BY:** A szerző nevét és a forrást meg kell említeni.
* **ND:** Nincsenek engedélyezett módosítások vagy átdolgozások. *[21]*

**CC BY-NC-ND (Creative Commons Attribution-NonCommercial-NoDerivatives)**

Ez a licenc a mű másolását és terjesztését kizárólag az eredeti formájában, csupán nem kereskedelmi célú felhasználása engedélyezett. A szerző nevét meg kell említeni. A CC BY-NC-ND licenc az alábbiakat tartalmazza:

* **BY:** A szerző nevét és a forrást meg kell említeni.
* **NC:** Csak nem kereskedelmi célú felhasználás engedélyezett.
* **ND:** Nincsenek engedélyezett módosítások vagy átdolgozások*. [21]*

# 8. Az elkészített program bemutatása

## 8.1 A játék eredete

Ahogy már a dolgozat címében is említettem az elkészített játék a klasszikus Space invaders játékot vette alapul, amely 1978-ban jelent meg a Taito kiadásában. A játék azonnal népszerűvé vált, két változatban: egy álló arcade szekrény és egy „koktélasztalos” szekrény formájában debütált. Az év végére több mint 100 000 gépet telepítettek belőle japánban, amelyek összesen 670 millió dollárt generáltak. A space invaders 1980-ban jelent meg Atari VCS-re és ez volt az első otthoni verzió és licencelt arcade cím konzolra. Ez a verzió drasztikusan megnövelte a játékkonzol eladásait, gyakorlatilag megnégyszerezte. A játékélmény arról szólt, hogy a játékosokat űrhajót irányítva harcolnak az egyre gyorsuló idegenekkel szemben, amely nagy mértékben vonzotta a játékosokat. Népszerűsége nyomán „gyilkos alkalmazásként” is emlegették a videojáték konzolok történetében. *[23]*

7.ábra: A space invaders játéktermekben: koktélasztalos és álló szekrények [23]

Tehát elmondható, hogy a space invaders hatalmas áttörést jelentett a videójáték-iparnak és a popkultúrának egyaránt. Az arcade játékokra való kereslet robbanásszerű növekedésnek indult, amelyet a játék sikere indított el. Nemcsak anyagi sikert hozott, hanem inspirációt is nyújtott a későbbi fejlesztők számára, megalapozva az arcade játékok térnyerését. *[23]*

## 8.2 A játék rövid bemutatása

A **Galactic Warrior** egy modern változata a klasszikus Space Invaders-típusú játékoknak, amely több szinten átívelő élményt nyújt. A játék angol nyelvű, szabályzata és részletes leírása közvetlenül a játéktér alatt érhető el, görgetéssel. Teljes mértékben böngészőkompatibilis, így különféle böngészőkön egyaránt fut, ezzel biztosítva, hogy széles körű felhasználói élményt nyújtson.

A játék központi témája az űr felfedezése, amely a sci-fi világának hangulatát idézi, és egyre erősebb ellenségekkel állítja szembe a játékost. A grafika az űr végtelenségét tükrözi, ahol különféle ellenséges hajók és lények próbálják megállítani a játékost. Ahogy haladunk előre, a pályák egyre nehezebbé válnak, és minden ötödik szintnél egy „boss fight” vár a játékosra. A zenei aláfestés dinamikusan alkalmazkodik az aktuális helyzethez: az egyszerűbb ellenségek legyőzésekor könnyedebb zene szól, míg a boss fight alatt intenzívebb dallamok fokozzák az izgalmakat.



8.ábra: Játékmenet szemléltetése

A program jelenlegi verziójában a játékos öt pályát teljesíthet, azonban a későbbiekben ez akár 30 szintre is bővíthető. A játék célja, hogy a játékos három élettel különböző ellenségekkel küzdjön meg az űr mélyén, minden ötödik szinten pedig egy különösen nehéz "boss fight" vár rá. Ezek az összecsapások nemcsak a játékos ügyességét, de a taktikai érzékét is próbára teszik. A későbbi bővítések során további boss fight-okkal és még izgalmasabb szintekkel találkozhat a játékos, valamint több lehulló bónusz is megjelenhet, amelyek még változatosabbá teszik a játékélményt és ösztönzik a folyamatos előrehaladást.

Egy játékbeli fizetőeszközt is bevezetésre került (angolul: in-game currency), amelyet a játékos a szintek teljesítése során gyűjthet. Ennek a fizetőeszköznek neve: rubin. A rubinok lehetőséget adnak egyedi pályahátterek és különleges képességekkel rendelkező űrhajók megvásárlására, ami erősebb fegyverzetet biztosít. Ez a rendszer hosszabb távon is motiválja a játékost a további játékra és a pályák újra játszásában, miközben javítja a játékélményt.

## 8.3 Programozási környezet ismertetése

A játék fejlesztéséhez Visual Studio Code (VS Code) integrált fejlesztői környezetet használtam, amely számos eszközzel megkönnyítette munkámat. A VS Code rugalmas és testreszabható, rengeteg bővítménnyel rendelkezik, így a feladat elkészítése során hatékonyan tudtam a kódban a formázást, hibakeresést elvégezni. A natív javascript nyelvet használtam a játék fejlesztéséhez, mert ez lehetővé tette, hogy közvetlen a böngészőben fussanak az interaktív elemek. Többek közt a felhasználó felületet, animációkat, a játékmenetet értem ezalatt. Nem használtam külső javascript keretrendszert, mint például: Phaser.js-t, mivel a natív javascript eszközkészlete elegendő volt arra, hogy a játékot és hozzá tartozó funkcióit hatékonyan valósítsam meg.

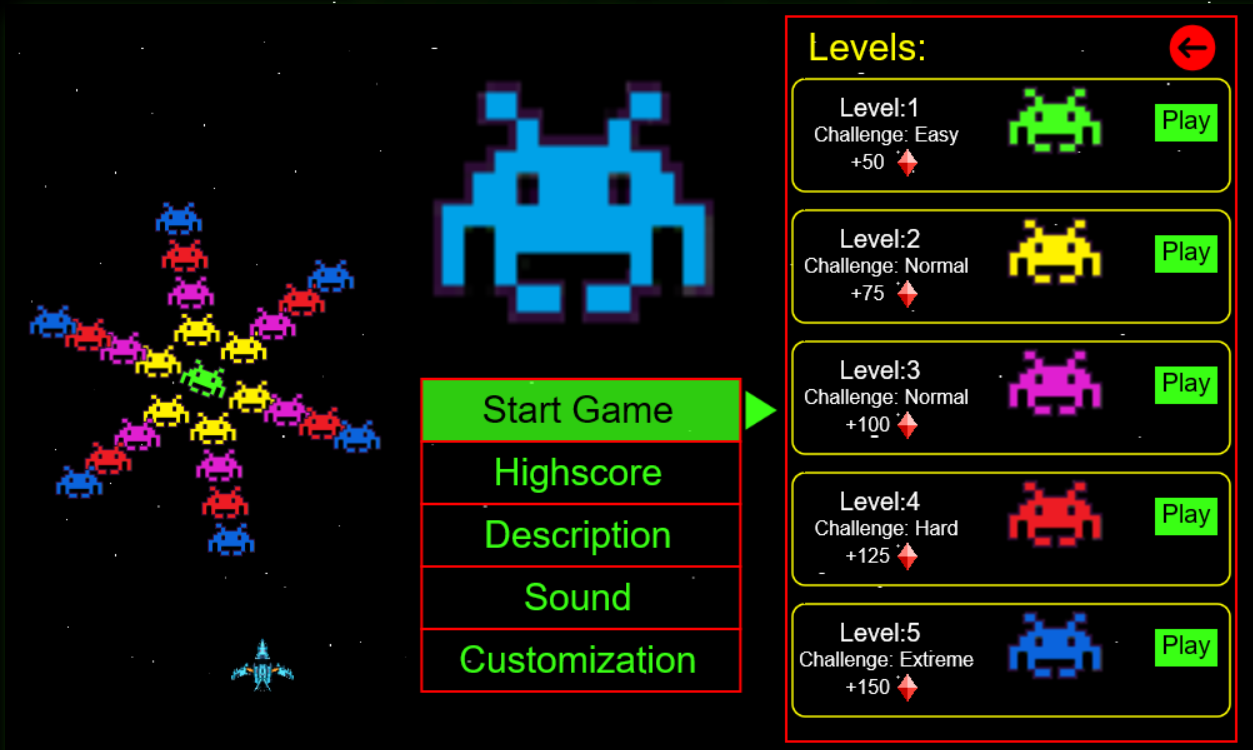
A játék különböző böngészőkben való tesztelése elengedhetetlen volt annak érdekében, hogy a lehető legszélesebb felhasználói kör számára legyen elérhető. Teszteltem opera, chrome, firefox, edge böngészőkön is, hogy ellenőrizzem a megfelelő kompatibilitást és a teljesítményt minden platformon. Ezzel a felhasználói élményt egységessé és hibamentessé alakítottam, függetlenül attól, hogy a felhasználó melyik böngészőt használja.

A játék grafikai elemeinek elkészítésekor többfajta képmanipuláló alkalmazásokat használtam, mint a gimp, paint és a photoshop főként a képek méretének módosításához, formázásához és olykor átrajzolásához vagy átszínezéséhez. Nem minden képi elemet én rajzoltam, néhányat játékfejlesztői oldalakról szereztem be CC0-ás licenccel, így ezek szabadon felhasználhatónak számítanak. A szoftverek biztosították számomra, hogy a képi elemek, mint a karakterek, hátterek és ikonok megfelelő méretben és formában jelenjenek meg a vásznon, így a felhasználók mindig tiszta és letisztult grafikai elemeket láthatnak.

## 8.4 A játék struktúrája

A Galactic Warrior felépítését tekintve 5 fő menüpont köré szerveződik, amelyek számos lehetőséget biztosítanak a játékosok számára, hogy elérjék a kívánt funkciókat, beállításokat.

1. **Start Game**: Ezzel a menüponttal megjelenik egy oldal panel, ahol a játék választhat magának szintet a képességeihez mérten. Itt elkezdheti az űr felfedezését és az ellenségek elleni harcot.
2. **Highscore**: A játék eredményeit rögzíti és jeleníti meg. Itt a játékos nyomon követheti saját teljesítményét, valamint javíthatja korábbi eredményeit, ezzel motiválva őt a még magasabb pontszámok elérésére.
3. **Description**: Ez a menüpont a Galactic Warrior játék tematikáját és lehetőségeit mutatja be.
4. **Sound**: A zenék és hangeffektek kezelése itt történik.
5. **Customization**: Ez a menüpont biztosítja a játék testreszabási lehetőségeit. A játékos saját ízlésének megfelelően változtathatja a vizuális elemeket, mint például a pályák háttereit vagy a játékban használt űrhajók megjelenését. Ez a funkció növeli a játék személyre szabhatóságát, lehetővé téve egyedi játékstílus kialakítását. Ebben a szekcióban költhetők el az összegyűjtött rubinok.



9.ábra: A játék menüje, választható szintekkel

Ez az öt fő menüpont egy jól átgondolt és logikus rendszert alkot, amely megkönnyíti a játékos számára a navigációt, a beállítások kezelését és a játékmenet maximális kihasználását. A felhasználói felület ezen elemei intuitívak, és mind a kezdők, mind a tapasztalt játékosok számára egyszerű hozzáférést biztosítanak a játék funkcióihoz és opcióihoz.

## 8.5 Képek kirajzolása

### 8.5.1 Egyszerűbb alakzatok, gombok rajzolása

Egyszerű alakzatok rajzolása a HTML5 canvas segítségével hatékonyan és könnyen kivitelezhető folyamat. A canvas beépített rajzfunkciói lehetővé teszik téglalapok, körök és más alapformák megjelenítését. Ezek az alakzatok nemcsak díszítőelemként szolgálhatnak, hanem interaktív elemekként is: akár gombokként, amelyekre szöveget is elhelyezhetünk. A téglalapok, keretek és szövegek kombinációjával könnyen létrehozhatunk funkcionális felhasználói felületeket.

drawMiniButton(x, y, width, height, text, color) {

    this.ctx.fillStyle = color;

    this.ctx.strokeStyle = "black";

    this.ctx.fillRect(x, y, width, height);

    this.ctx.strokeRect(x, y, width, height);

    this.ctx.fillStyle = "white";

    this.ctx.fillText(text, x + width / 2, y + height / 2 + 5);

}

A *drawMiniButton()* függvény egy egyszerű módszert kínál gombok rajzolására. A paraméterek meghatározzák a gomb helyét (x, y), méretét (width, height), a gombon megjelenő szöveget és a színt. Először a *fillRect()* segítségével egy színes téglalapot rajzolunk a kívánt koordinátákra, majd a *strokeRect()* funkcióval egy fekete keretet adunk hozzá, amely kiemeli a gombot a háttérből. Végül a gombra kerül a szöveg, amit a *fillText()* metódussal rajzolunk ki, a gomb szélességéhez igazított pozícióban, hogy középen jelenjen meg. A kirajzolása pedig a következőképp néz ki:

draw() {

const buttonWidth = 50;

const buttonHeight = 20;

this.drawMiniButton(20, 445, buttonWidth, buttonHeight, "SELECT", "#1b71c2");

}

A gomb szélessége (*buttonWidth* = 50) és magassága (*buttonHeight* = 20) előre definiált változók. A függvény paraméterei a gomb pozícióját (x = 20, y = 445), méretét, a rajta megjelenő szöveget ("SELECT") és a színét ("#1b71c2") határozzák meg. Ez a hívás egy kék színű, „SELECT” feliratú gombot rajzol a megadott koordinátákra a canvas elemen:



10.ábra: Select gomb megrajzolása

### 8.5.2 Képek kirajzolása a játékban (enemy, player, boss)

Az ellenségek kirajzolását az *enemy.js* osztály végzi, ahol az ellenség pozícióját, méretét és megjelenését dinamikusan határozzuk meg. Az ellenség sprite-ját a *this.image.src = 'images/enemy/enemy${imageNumber}.png'* sor tölti be, amely az ellenség azonosítójától függően változik. Erre azért van szükség, hogy több képet fel tudjak használni a különböző ellenségek megrajzolásához. A pozíciót az x és y koordináták határozzák meg, míg a mérete fixen 60x35 pixel. Az ellenségek értéke (pontok számításához) az imageNumber változó alapján kerül meghatározásra. A *draw()* függvény minden képkockánál kirajzolja az ellenséget a megadott koordinátákra a *ctx.drawImage()* segítségével.

    constructor(x,y,imageNumber){

        this.x = x;

        this.y = y;

        this.width = 60;

        this.height = 35;

        this.value = imageNumber;

        this.image = new Image();

        this.image.src = `images/enemy/enemy${imageNumber}.png`;

    }

    draw(ctx){

        ctx.drawImage(this.image,this.x,this.y,this.width,this.height);

    }

Az ellenségek létrehozása egy mátrix (*enemyMap*) alapján történik, amely meghatározza az ellenségek típusát és elhelyezkedését. Minden sor és oszlop egy adott ellenségtípust jelöl, a nulla pedig azt, hogy azon a helyen nincs ellenség.

    enemyMap = [

        [3, 3, 3, 1, 3, 3, 2, 3, 3, 3],

        [5, 0, 6, 0, 0, 0, 0, 6, 0, 5],

        [4, 0, 0, 0, 5, 5, 0, 0, 0, 4],

        [3, 0, 0, 0, 0, 0, 0, 0, 0, 3],

        [2, 0, 0, 0, 0, 0, 0, 0, 0, 2],

        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

    ];

A *createEnemies()* függvény végigmegy a mátrixon, és az ellenségeket a megfelelő koordinátákra helyezi, az indexek alapján számolva a pozíciókat. A *drawEnemies()* függvény minden ellenséget kirajzol és mozgat a megadott sebességgel.

    drawEnemies(ctx) {

        this.enemyRows.flat().forEach((enemy) => {

            if (enemy) {

                enemy.move(this.xVelocity, this.yVelocity);

                enemy.draw(ctx);

            }

        });

    }

    createEnemies() {

        this.enemyMap.forEach((row, rowIndex) => {

            this.enemyRows[rowIndex] = [];

            row.forEach((enemyNumber, enemyIndex) => {

                if (enemyNumber > 0) {

                   this.enemyRows[rowIndex].push(

                   new Enemy(enemyIndex \* 45, rowIndex \* 30 + 30, enemyNumber)

                  );

                }

            });

        });

    }

A játékos (Player) és a főellenség (Boss) rajzolása is hasonló logikán alapul, mint az ellenségek esetében, ám a paraméterek és a funkciók terén némi eltérés figyelhető meg. A fő különbség az, hogy fix képeket használok. Míg az ellenségek elhelyezkedését egy előre meghatározott mátrix alapján állítom be, a játékos és a főellenség közvetlenül a kívánt koordinátákra kerülnek. Ezen kívül különböző lövéstípusok, mozgások és egyéb funkciók is kezelésre kerülnek ezen 2 elem kirajzolásakor.

### 8.5.3 Animáció

A rubin animációját a játékban sprite alapú technikával valósítottam meg. A sprite technika lényege, hogy egy nagyobb képből több kisebb képkockát nyerünk ki, amelyek egymás utáni gyors megjelenítésével hozunk létre egy mozgás illúziót. Ebben az esetben a rubin animációja egyetlen sorban elhelyezett 7 képkockából áll, és minden egyes képkocka a rubin egy másik pozícióját ábrázolja. Az alábbi képpel szemléltetem, hogy milyen képkockákat használ a sprite sheet, amelyek alapján a rubin animációja létrejön:



11.ábra: rubin animációs képkockák

Az animáció létrehozását a következőképpen valósítottam meg:

animateRuby(xPos, yPos, scale) {

    const spriteWidth = 24;

    const spriteHeight = 24;

    let frameIndex = 0;

    let count = 0;

    const spriteSheet = new Image();

    spriteSheet.src = "images/ikon/ruby.png";

    const animate = () => {

        this.ctx.drawImage(

            spriteSheet,

            frameIndex \* spriteWidth,

            0,

            spriteWidth,

            spriteHeight,

            xPos,

            yPos,

            spriteWidth \* scale,

            spriteHeight \* scale

        );

        count++;

        if (count >= 16) {

            frameIndex++;

            count = 0;

        }

        if (frameIndex > 6) {

             frameIndex = 0;

        }

    };

    const frame = () => {

        if (this.animationRun) {

            animate();

            requestAnimationFrame(frame);

        }

    };

    spriteSheet.onload = () => {

        this.animationRun = true;

        frame();

    };

}

A *spriteWidth* és a *spriteHeight* változók határozzák meg, hogy egy képkocka milyen széles és magas. Jelen esetben ez 24x24 pixel méretű. A sprite felhasznált forrásképe a *"images/ikon/ruby.png"*, amit az *Image()* objektum segítségével töltök be. A *frameIndex* változó tartja nyilván, hogy jelenleg melyik képkocka van éppen kirajzolva. A képkockák sorban következnek egymás után. Az *xPos* és *yPos* adja meg, hogy a rubin animáció hol jelenik meg a képernyőn, a *scale* pedig a képkockák méretarányos nagyítását vagy kicsinyítését szabályozza. A *count* változó felelős azért, hogy az animáció mennyire legyen gyors vagy lassú. Minden alkalommal, amikor az animáció fut, a *count* növekszik. Ha eléri a 16-ot, a *frameIndex* eggyel nő, így a következő képkocka kerül kirajzolásra. Ez a mechanizmus biztosítja, hogy az animáció ne legyen túl gyors, és egy sima, egyenletes mozgás jöjjön létre. A rubin animáció 7 képkockából áll (0-tól 6-ig indexelve), ezért amikor a *frameIndex* értéke eléri a 6-ot, visszaállítjuk 0-ra, ezzel az animáció újraindul az első képkockáról. Így a rubin animációja folyamatosan ismétlődik. A *requestAnimationFrame()* függvény hívása felelős azért, hogy az animáció minden képkockát megfelelő időzítéssel kirajzoljon.

Az animáció akkor indul el, amikor a sprite sheet betöltődik. Ezt az *onload* esemény figyeli, és amikor a kép teljesen készen áll, beindul az animáció a *frame()* függvény segítségével.

A rubin animációját a következő sorral rajzolom ki, ahol megadtam a rubin kirajzolási helyét (xPos: 50, yPos: 390) és a méretbeli értéket (scale: 1).

this.animateRuby(50, 390, 1);

## 8.6 Játéklogika

A játék indítása és a szint kiválasztása egyszerű és letisztult folyamat. Először is a ***startGame()*** funkcióval indul a játék. Ez leállít minden játszódó zenét, majd beállítja a háttérképet a szinthez. A felhasználó választása szerint a választott pálya elindul, amit egy animációs ciklus: *RequestAnimationFrame()* garantál, ezzel így a játék állandóan frissülni fog.

function startGame(level) {

    stopAllMusic();

    gameBackground.src = costumization.field;

    previousTime = performance.now(); // kezdeti idő beállítása

    gameRunning[level - 1] = true;

    requestAnimationFrame(currentTime => gameLoop(currentTime, level));

}

A játék menetét a ***gameLoop()*** funkció kezeli. Ez azért is felelős, hogy a játék megfelelő sebességgel fusson és frissítések mindig az időzítésnek megfelelően történjenek meg. A *gameLoop()* időnként újból lefut és az aktuális szint szerint frissül a játék. Ebben a metódusban dönti el a program, hogy éppen melyik szint fut és utána az adott szinthez tartozó események futása következik.

function gameLoop(currentTime, level) {

    if (!gameRunning[level - 1]) return;

    let delta = currentTime - previousTime;

    if (delta > interval) {

        previousTime = currentTime - (delta % interval);

        switch (level) {

            case 1: // level 1

                gameHandler(1, enemyHandler1);

                break;

            case 2:

                gameHandler(2, enemyHandler2);

                break;

            case 3:

                gameHandler(3, enemyHandler3);

                break;

            case 4:

                gameHandler(4, enemyHandler4);

                break;

            case 5:

                gameHandler(5, enemyHandler5);

                break;

        }

    }

animationFrameId = requestAnimationFrame(currentTime => gameLoop(currentTime, level));

}

A szinteken az eseményeket a ***gameHandler()*** kezeli. Ez a funkció olyan dolgokat szabályoz, mint a zene lejátszása, az ellenségek mozgása, valamint a játékos pontszáma, élete és eltöltött ideje a pályán. Ha a játékos az ötödik szinten van, ahol szembe kell néznie a fő ellenséggel, akkor a szokásos zene helyett egy speciális, a szinthez méltó dal szólal meg. Ez a funkció ezen felül figyeli a játékosok és az ellenségek közötti ütközéseket is (*CollideWithObjects()*), és ellenőrzi, hogy mikor van vége a játéknak: *checkGameOver(level)*. A játék automatikusan leáll, ha a játékos minden életet veszít, vagy teljesíti a pályát. A zene elhallgat és a játék véget ér. Ha azonban a játék még mindig folyamatban van, minden elem, mint például a pontszám, az idő és az életek újra frissül a következő képkockán.

function gameHandler(level, enemyHandler) {

    let soundToUse = gameSound;

    // Boss fight esetén más zene

    if (level === 5) {

        soundToUse = bossSound;

    }

    checkGameOver(level);

    collideWithObject();

    ctx.drawImage(gameBackground, 0, 0, canvas.width, canvas.height);

    displayGameOver(level);

    if (!isGameOver) {

        if (enemyHandler.soundEnabled) {

            if (soundToUse.paused) {

                soundToUse.play();

            }

        } else {

            if (!soundToUse.paused) {

                soundToUse.pause();

            }

        }

        settings(level);

        ctx.fillStyle = "yellow";

        ctx.font = "20px sans-serif";

        ctx.fillText("Score: " + enemyHandler.score, 125, 23);

        if (scoreImage.complete) {

            ctx.drawImage(scoreImage, 185, 2, 25, 25);

        }

        ctx.fillStyle = "white";

        ctx.font = "20px sans-serif";

        ctx.fillText("Time: " + updateTime(), canvas.width / 2, 23);

        if (timerImage.complete) {

            ctx.drawImage(timerImage, canvas.width / 2 + 60, 2, 25, 25);

        }

        ctx.fillStyle = "red";

        ctx.font = "20px sans-serif";

        ctx.fillText("Life: " + life, canvas.width - 125, 23);

        if (heartImage.complete) {

            ctx.drawImage(heartImage, canvas.width - 95, 2, 25, 25);

        }

        document.addEventListener('keydown', function (event) {

            if (event.key === 'Escape') {

                event.preventDefault();

            }

        });

    } else {

        gameRunning[level - 1] = false;

        soundToUse.pause();

    }

}

A ***settings()*** funkció a játék különböző osztályaiból származó beállításokat kezeli. A funkció biztosítja, hogy a ***Costumization*** osztályban megadott vizuális elemek, mint a játékos űrhajójának megjelenése, a lövedékek színe a játékmenet során megjelenjenek. A ***Sound***osztályban kiválasztott hangbeállításokat is figyelembe vesszük, hiszen a játékos által beállított hangerő és a hang engedélyezése megfelelően lesz kezelve. Ezzel a funkcióval, tehát a játékos által kiválasztott vizuális elemek és hangokat állítom be.

function settings(level) {

    player.image.src = costumization.selectedPlayer;

    player.draw(ctx);

    playerBulletController.draw(ctx);

    enemyBulletController.draw(ctx);

    bossBulletController.draw(ctx);

    giftController.draw(ctx);

    gift2Controller.draw(ctx);

    meteorController.draw(ctx);

    const enemyHandler = getEnemyHandler(level);

    if (enemyHandler) {

        enemyHandler.draw(ctx);

        enemyHandler.soundEnabled = sound.soundOn;

        enemyHandler.enemyDeathSound.volume = (sound.volume / 100);

    }

    // hangerő beállítása

    playerBulletController.soundEnabled = sound.soundOn;

    enemyBulletController.soundEnabled = sound.soundOn;

    bossBulletController.soundEnabled = sound.soundOn;

    meteorController.soundEnabled = sound.soundOn;

    playerBulletController.shootSound.volume = (sound.volume / 100);

    enemyBulletController.shootSound.volume = (sound.volume / 100);

    bossBulletController.shootSound.volume = (sound.volume / 100);

    meteorController.meteorBoom.volume = (sound.volume / 100);

    // Lövedék színének beállítása

    playerBulletController.bulletColor = costumization.playerBulletColor;

    enemyBulletController.bulletColor = costumization.enemyBulletColor;

    bossBulletController.bulletColor = costumization.enemyBulletColor;

}

A ***displayGameOver()*** a játék végén egyértelműen jelzi, hogy a játékos teljesítette vagy sem az adott szintet. Az i*sGameOver* változó segítségével határozom meg, hogy a játéknak vége van-e, ennek megfelelően jelenik meg a „You Win” vagy a „Game Over” felirat. A megmaradt életek számából további pontok kerülnek kiszámításra és hozzáadódnak az pontszámhoz, majd ezt is megjelenítjük a képernyőn a játékkal eltöltött idővel együtt. Az *enemyHandler* használatával az elért pontszám mentésre kerül és hogy teljesített vagy sem a játszott szint. Ez mellett a játékbeli fizetőeszköz (rubin) is itt kerül jóváírásra a pálya teljesítése esetén. Ezekről részletesen később. Ezentúl a játékosnak lehetősége van visszatérni a menübe [ESCAPE] billentyűvel vagy újrakezdeni a játékot az R lenyomásával.

function displayGameOver(level) {

    if (isGameOver) {

        let text = didwin ? "You win" : "Game Over";

        let addScore = life \* 50; // remaining life added to score

        ctx.fillStyle = "#39ff14";

        ctx.font = "30px Arial";

        const enemyHandler = getEnemyHandler(level);

        if (enemyHandler) {

       enemyHandler.score += addScore;

       ctx.fillText("Total Score: " + enemyHandler.score, canvas.width / 2, 380);

       let levelComplete = didwin ? true : false;

       highscore.setHighscore(level, enemyHandler.score, levelComplete);

       const rubinBonus = level \* 25 + 25;

            if (levelComplete) {

                increaseRubin(rubinBonus);

            }

        }

        drawButtonBack();

// kirajzolások

    ...

        // Gomb lenyomás figyelő

        const keyDownHandler = function(event) {

            if (event.key === "Escape") {

                isStartButtonActive = false;

                removeListeners();

                menu();

            } else if (event.key === "r") {

                removeListeners();

                restartGame(level);

            }

        };

// eseménykezelők meghívása, hozzáadása

...

    }

}

Az újraindításért a ***RestartGame()*** funkció felelős, amely először megszakítja az animációt a letárolt ID alapján: *cancelAnimationFrame(animationFrameId)*, majd alaphelyzetbe állítja a játékban használt változókat, többek közt az időt, életek számát, játékos pozícióját. De ez mellett kiüríti a pályán a lövedékeket, pl: *playerBulletController.clearBullets()*, meteorokat: *meteorController.clearMeteors()* és számos más elemet. Ezután az ellenségek újbóli létrehozása következik az EnemyHandler-rel és újraindítja a zenét, hogy a játék visszatérjen a kiválasztott szinthez, és folytatódhasson a játékélmény.

function restartGame(level) {

    if (animationFrameId !== null) {

        cancelAnimationFrame(animationFrameId);

        animationFrameId = null; // animáció ID nullázása

    }

    gameRunning = [false, false, false, false, false];

    seconds = 0;

    isGameOver = false;

    didwin = false;

    life = 3;

    player.x = canvas.width / 2;

    player.y = canvas.height - 75;

    enemyBulletController.clearBullets();

    playerBulletController.clearBullets();

    bossBulletController.clearBullets();

    giftController.clearGifts();

    gift2Controller.clearGifts();

    meteorController.clearMeteors();

    const enemyHandler = getEnemyHandler(level);

    if (enemyHandler) {

        enemyHandler.score = 0;

        enemyHandler.createEnemies();

        enemyHandler.resetGame();

        if (enemyHandler.soundEnabled) {

            gameSound.currentTime = 0;

        }

        if (level === 5) {

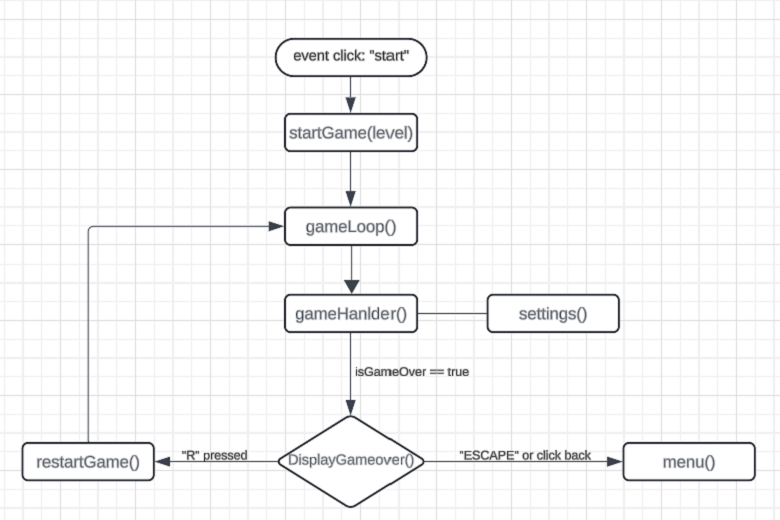
            bossSound.currentTime = 0;

        }

        startGame(level);

    }

}



12.ábra: Játékmenet folyamatábra

Most, hogy sikerült átbeszélni a játék vezérlését, térjünk át az ellenségeket kezelő osztály, az ***EnemyHandler*** bemutatására. Az ***EnemyHandler*** osztály felelős a játékban megjelenő ellenségek kirajzolásáért, mozgásáért és irányításáért. Az osztály kezeli az ellenségek helyzetét, sebességét, lövési mechanizmusát, valamint azokat az eseményeket, amikor az ellenségeket eltalálja a jákétkos. A kód több különálló, ám egymással szorosan együttműködő részre bontható. Az osztály alapvető tulajdonságai között találjuk az *enemyMap*, amely egy mátrix formájában határozza meg az ellenségek elrendezését és típusát. Minden szám egy adott ellenségtípust jelöl, amelyet később az *enemyRows* tömbben tárolunk el. Ez a tömb tartalmazza az aktuálisan játékban lévő ellenségeket, amelyek folyamatosan frissülnek, ahogy az ellenségek megsemmisülnek vagy újra létrejönnek. A mozgás irányát és sebességét a *currentDirection, xVelocity* és *yVelocity* változók határozzák meg. Ezek a változók hatására az ellenségek megfelelően mozognak a képernyőn. Kezdetben a mozgás iránya lefelé van beállítva (*MovingDirection.down*), ami a későbbiekben módosul a játékállapot alapján. Az alapértelmezett sebességeket az *xVelocity* és *yVelocity* változók határozzák meg, amelyeket a *defaultXVelocity* és *defaultYVelocity* változók szabályoznak.

    enemyMap = [. . .]; // mátrix feltöltése ellenségekkel

    enemyRows = [];

    currentDirection = MovingDirection.down;

    xVelocity = 0;

    yVelocity = 0;

    defaultXVelocity = 1;

    defaultYVelocity = 1.5;

    fireBulletTimerDefault = 70;

    fireBulletTimer = this.fireBulletTimerDefault;

    dropGiftTimer = 700;

    dropMeteorTimer = 300;

    score = 0;

A *draw()* metódus a kulcsfontosságú funkció az ellenségek kirajzolásához. Ez a metódus először a mozgás időzítését kezeli a *decrementMoveDownTimer()* segítségével, majd frissíti az ellenségek sebességét és irányát az *updateVelocityAndDirection()* függvényen keresztül. Miután az irány és sebesség beállítása megtörtént, a *collisionDetection()* metódus ellenőrzi, hogy az ellenségeket találat érte-e a játékos lövedékeivel. Ha egy ellenséget eltalálnak, akkor az eltávolításra kerül, és a játékos pontszáma növekszik az *addScore()* függvényen keresztül.

    draw(ctx) {

        this.decrementMoveDownTimer();

        this.updateVelocityAndDirection();

        this.collisionDetection();

        this.drawEnemies(ctx);

        this.resetMoveDownTimer();

        this.fireBullet();

        this.dropGift();

        this.dropMeteor();

    }

Az ellenségek lövését a *fireBullet()* metódus kezeli. Ez a metódus folyamatosan csökkenti a *fireBulletTimer* értékét, és amikor ez az időzítő eléri a nullát, egy véletlenszerűen kiválasztott ellenség tüzet nyit a játékosra. A lövést az *enemyBulletController* irányítja, amely az ellenség pozíciójából lövi ki a lövedéket. Ez az elem növeli a játékmenet dinamikáját, mivel a játékosnak folyamatosan mozognia kell, hogy elkerülje a lövedékeket:

    fireBullet() {

        this.fireBulletTimer--;

        if (this.fireBulletTimer <= 0) {

            this.fireBulletTimer = this.fireBulletTimerDefault;

            const allEnemies = this.enemyRows.flat();

            const enemyIndex = Math.floor(Math.random() \* allEnemies.length);

            const enemy = allEnemies[enemyIndex];

         this.enemyBulletController.shoot(enemy.x + enemy.width / 2, enemy.y, -3);

        }

    }

Az ***updateVelocityAndDirection()*** metódus felel az ellenségek mozgásának frissítéséért és irányuk dinamikus változtatásáért. A függvény először kiszámítja az ellenségek szélső pozícióit: a legalsó (***bottomMostY***) és legfelső (***topMostY***) pontokat, valamint a legjobban jobbra és balra lévő ellenségeket (***rightMostX, leftMostX***). Ezek az értékek meghatározzák, hogy az ellenségek milyen mozgási irányt vesznek fel a képernyő szélei és teteje alapján. Ha például az ellenségek elérik a képernyő alját, akkor felfelé fognak mozogni, míg ha elérik a jobb szélt, balra váltanak.

Az alábbi kódrészlet mutatja az ellenségek mozgásának irányát kezelő logikát:

if (this.currentDirection === MovingDirection.down) {

                this.xVelocity = 0;

                this.yVelocity = this.defaultYVelocity;

                if (bottomMostY + this.defaultYVelocity >= this.canvas.height) {

                    this.currentDirection = MovingDirection.up;

                    break;

                }

            } else if (this.currentDirection === MovingDirection.up) {

                this.xVelocity = 0;

                this.yVelocity = -this.defaultYVelocity;

                if (topMostY - this.defaultYVelocity <= 0) {

                    this.currentDirection = (rightMostX >= this.canvas.width) ? MovingDirection.left : MovingDirection.right;

                    break;

                }

A mozgás iránya annak függvényében módosul, hogy az ellenségek elérik-e a képernyő szélét vagy alját. Például, ha a ***bottomMostY*** értéke meghaladja a képernyő magasságát, az irány felfelé változik (***MovingDirection.up***), míg a ***rightMostX*** érték meghatározza, hogy jobbról balra kell-e váltani. Az utolsó rész a ciklusok közötti irányváltás logikáját kezeli, például amikor balról jobbra vagy jobbról balra váltanak az ellenségek:

else if (this.currentDirection === MovingDirection.right) {

...// jobbra mozgás logikája

}

else if (this.currentDirection === MovingDirection.left) {

... // balra mozgás logikája

}

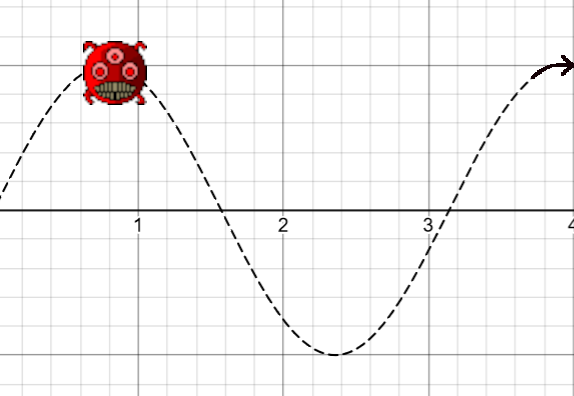
A ciklus végén az ***enemyRows.flat()*** függvény az ellenségek tömbjét lapítja le egy sík tömbbé, és mindegyik ellenséget a megfelelő **xVelocity** és **yVelocity** sebességekkel mozgatja:

this.enemyRows.flat().forEach((enemy) => {

            enemy.move(this.xVelocity, this.yVelocity); }

);

Ezen felül a boss/főellenség esetében bevezettem, hogy ne csupán egyenes mozgást végezzen, hanem hullámzó mozgást kövessen, amit matematikai függvények és fizikai alapelvek bevonásával valósítottam meg. A megírt mozgásforma különbözik az egyszerű egyenes vonalú mozgástól, mivel a főellenség pozícióját egy szinusz függvény alapján dinamikusan változtatja. Ezzel az újítással a cél az volt, hogy a mozgás látványosabb legyen, ezzel is növelve a játékélményt.

A hullámzó mozgást a *waveMove()* metódus vezérli, amely folyamatosan változtatja az ellenség elhelyezkedését a képernyőn, miközben halad előre. A konstruktorban 3 új változó kerül bevezetésre: a *radiu*s, amely a hullám amplitúdóját, azaz függőleges kilengését szabályozza. A *speed*, mely a hullámzás gyorsaságáért felel és az *angle*, ami a hullámzás pillanatnyi szögét jelzi. Ahogy a *waveMove()* metódus fut, az *angle* értéke folyamatosan növekszik a *speed* hatására, ezzel a boss folyamatos hullámzó mozgást tud végezni a képernyőn. Tehát az y pozíciót a szinuszfüggvény értékei alapján változtatja, ezzel fel-le mozgást végezve, míg x irányba lineárisan halad tovább.

    constructor(...) {

        ... // főellenség kirajzolása

        this.radius = 1;

        this.speed = 0.02;

        this.angle = 1;

    }

    // Hullámzó mozgás

    waveMove(xVelocity, yVelocity) {

        this.x += xVelocity;

        this.angle += this.speed;

       const circularY = this.radius \* Math.sin(this.angle);

        this.y += yVelocity + circularY;

13.ábra: Főellenség mozgásának szemléltetése

    }

Végül, az EnemyHandler osztályok olyan extra funkciókat valósítanak meg, mint az ajándékok ledobása (*dropGift*) és meteoritok generálása (*dropMeteor*). Ezek az események növelik a játékmenetben a változatosságot és izgalmakat. Ezek a funkciók időzítetten aktiválódnak a *dropGiftTimer* és *dropMeteorTimer* változók alapján, amelyek a játék során folyamatosan csökkennek, amíg el nem érik a nullát. A leeső elemkre példa:

    dropMeteor() {

      this.dropMeteorTimer--;

      if (this.dropMeteorTimer <= 0) {

       this.dropMeteorTimer = Math.floor(Math.random() \* (1200 - 600)) + 600;

       const x = Math.random() \* this.canvas.width;

      const y = 0;

       const Yvelocity = -4;

       const timeTillNextMeteorAllowed = 10;

       this.meteorController.drop(x, y, Yvelocity, timeTillNextMeteorAllowed);

      }

    }

## 8.7 Canvas eseménykezelés

### 8.7.1 Billentyűlenyomás

A kódban a billentyűlenyomás kezelését a *handleKeyDown* segítségével oldom meg, amely a keydown eseményekre reagál. Az *event.code* alapján határozom meg, hogy melyik billentyűt nyomták le és ennek megfelelően állítom be a logikai változókat. Például a jobbra és a bara mozgást (*ArrowRight, ArrowLeft*) esetén a mozgást *moveRight* és *moveLeft* változók jelzik.

A szóköz, avagy a space lenyomásakor először is megelőzöm az alapvető funkciót, ami az oldal aljára ugrás lenne: *event.preventDefault()*. Majd a lövés változót igaz értékre állítom: *this.shootPressed=true*, tehát ennek hatására tüzel a játékos. Ehhez hasonlóan blokkolom a fel és le nyilak alapértelmezett viselkedését, hogy ne görgessék az oldalt.

    handleKeyDown = (event) => {

        switch (event.code) {

            case "ArrowRight":

                this.moveRight = true;

                break;

            case "ArrowLeft":

                this.moveLeft = true;

                break;

            case "Space":

                event.preventDefault();

                this.shootPressed = true;

                break;

            case "ArrowUp":

                event.preventDefault();

                break;

            case "ArrowDown":

                event.preventDefault();

            default:

                break;

        }

    };

Majd hozzáadtam az eseménykezelőt a következő módon:

    document.addEventListener("keydown", this.handleKeyDown);

### 8.7.2 Kattintás

A kattintások kezelésére egy *clickHandler* nevű függvényt hoztam létre, amely a canvas-en végrehajtott kattintásokra reagál. A kattintás helyének meghatározásához a vászon pozícióját veszem figyelembe az oldalhoz viszonyítva: *getBoundingClinetRect()*. Ezt követően az egérkattintás koordinátái alapján döntöm el, hogy a felhasználó melyik interaktív elemre kattintott, és ennek megfelelően hajtom végre a kívánt műveletet.

Példaként a Sound menüpontot mutatom be. Ha a felhasználó a hangerő növelésre vagy csökkentésére kijelölt területekre (gombokra) kattint, akkor a progam a hangerő növelését vagy csökkentését végzi el, majd a vászon újrarajzolásával frisssíti a vizuális elemeket. Erre azért van szükség, hogy a felhasználó a hangerő megváltoztatásnak hatására azonnal kész információkkal rendelkezzen a hangerő aktuális állapotáról. A hang ki- és bekapcsolása is hasonló módon működik: a kattintás pozíciója alapján történik a művelet aktiválása, amely szintén megjelenik a frissített felületen.

A használt click eseménykezelés nem csak itt használatos, hanem a többi menüelemnél is, ahol interaktív funkciókra van szükség a játékban. A *clickHandler* így általánosan használható a különböző menüpontok kezelésére, biztosítva a felhasználói kattintások megfelelő feldolgozását és a vizuális elemek frissítését.

const clickHandler = (event) => {

            const rect = this.canvas.getBoundingClientRect();

            const x = event.clientX - rect.left;

            const y = event.clientY - rect.top;

 if (x >= 910 && x <= 974 && y >= 10 && y <= 74 && this.isMenuActive === true) {

                this.menuCallback(); // menu() visszahívása

                this.isMenuActive = false;

            }

 if (x >= 350 && x <= 400 && y >= 180 && y <= 230 && this.isMenuActive === true) {

                this.toggleSound(); // Hang ki/be kapcsolása

                this.draw(); // gomb újrarajzolása, on -> off / off -> on váltás

            }

if (x >= 750 && x <= 800 && y >= 150 && y <= 200 && this.isMenuActive === true) {

                this.increaseVolume();

                this.draw();

            }

if (x >= 750 && x <= 800 && y >= 220 && y <= 270 && this.isMenuActive === true) {

                this.decreaseVolume();

                this.draw();

            }

};

Az eseménykezelőt a *this.canvas.addEventListener*(*'click',clickHandler*); sorral adom hozzá a canvas-hez, hogy minden egyes kattintásra automatikusan lefusson a megfelelő művelet.

 this.canvas.addEventListener('click',clickHandler);

### 8.7.3 Egérmozgás

const mouseMoveHandler = (event) => {

    if (isStartButtonActive) return;

    const rect = canvas.getBoundingClientRect();

    const x = event.clientX - rect.left;

    const y = event.clientY - rect.top;

    ctx.drawImage(background, 0, 0, canvas.width, canvas.height);

    ctx.drawImage(logoImage, canvas.width / 3 - logoImage.width / 2, canvas.height / 4 - logoImage.height / 2 - 100, 350, 300);

    for (const button of buttons) {

        if (x >= button.x && x <= button.x + button.width && y >= button.y && y <= button.y + button.height) {

            ctx.fillStyle = "rgba(57, 255, 20, 0.8)"; // neon green

            ctx.fillRect(button.x, button.y, button.width, button.height);

            ctx.strokeRect(button.x, button.y, button.width, button.height);

            ctx.fillStyle = "#000000"; // black

            ctx.font = "30px sans-serif";

            ctx.fillText(button.text, button.x + 125, button.y + button.height / 2 + 10);

        } else {

            makeButton(button.text, button.x, button.y, button.width, button.height);

        }

    }

};

canvas.addEventListener('mousemove', mouseMoveHandler);

A fenti kódrészletben egy egérmozgás kezelő van megvalósítva, ami a vásznon bekövetkezett mozgásra reagál. Az egér aktuális pozícióját a *canvas.getBoundingClientRect()* metódussal számítja ki, majd ez alapján állapítja meg, hogy az egér valamelyik gomb területén van-e éppen. Ha igen, akkor a gomb háttérszíne megváltozik zöldre (*ctx.fillStyle = "rgba(57, 255, 20, 0.8)"* ) és a betűszín pedig feketére vált (*ctx.fillStyle = "#000000"*), ezzel jelezve a felhasználónak, hogy a gomb interaktív. Ha az egér nem a gombon helyezkedik el, akkor a *makeButton()* függvény visszaállítja a gomb megjelenését az eredeti helyzetbe. A háttér és a logó újrarajzolása minden egérmozgásnál megtörténik, hogy a felület friss maradjon. Továbbá a handler nem hajt végre műveletet, ha a start gomb aktív (*isStartButtonActive*), ami a játék során egy állapotjelzés. Erre azért van szükség, mivel a játékmenet elindásakor problémát okozhat, ha elmozdítja a felhasználó az egeret. Az eseménykezelőt *a canvas.addEventListener('mousemove', mouseMoveHandler)* sor kapcsolja össze az egérmozgással, így az interakciók dinamikusan reagálnak a mozdulatokra.

## 8.8 Adatok tárolása

Az adatok tárolását a webalkalmazásoknál gyakran használat localStorage segítségével oldottam meg, ami lehetővé teszi, hogy letároljam a játékos adatait, ha bezárja a böngészőt vagy újraindítja a számítógépet. Esetemben localStorage-ban kulcs-érték párokat használok, ahol a kulcs a szint és az érték a *highscores* objektum, amely a *score*-t (pontszámot) és a *complete* (teljesítés állapota) változókat tárolja. Ennek köszönhetően a játékos megtudja tekinteni korábbi teljesítményeit és akár ösztönözheti őt jobb eredmények elérésére is. Minden szinthez elmentésre kerül a legmagasabb elért pontszám valamint hogy a szint teljesítve lett-e. Ezt a *didwin* változó felvett értéke alapján határoztam meg.

let levelComplete = didwin ? true : false;

highscore.setHighscore(level, enemyHandler.score, levelComplete);

A *Highscore.js* osztályban a *setHighscore()* metódus felelős az eredmények eltárolásáért. Itt ellenőrzöm, hogy a szinthez tartozó pontszám meghaladja-e a korábbit. Ha igen frissítem a pontszámot és a szint teljesítésének állapotát. Ezután a *localStorage.setItem()* metódussal az eredmények elmentésre kerülnek a localStorage-ba, hogy az eredmények később is elérhetőek legyenek.

setHighscore(level, score, complete) {

    if (level >= 1 && level <= 5) {

        if (score > this.highscores[level].score) {

            this.highscores[level].score = score;

            this.highscores[level].complete = complete;

            // Adatok mentése a localStorage-ba

            localStorage.setItem('highscores', JSON.stringify(this.highscores));

        }

    }

}

Az osztály konstruktorába először betöltöm localStorage-ban tárolt eredményeket, ha léteznek, különben alapértelmezett értékekkel inicializálom a szintekhez tartozó adatokat: minden szint 0 pont és a teljesítés állapota hamis értéket kap.

const savedScores = localStorage.getItem('highscores');

        if (savedScores) {

            this.highscores = JSON.parse(savedScores);

        } else {

            this.highscores = {

                1: { score: 0, complete: false }, // Level 1

                2: { score: 0, complete: false },

                3: { score: 0, complete: false },

                4: { score: 0, complete: false },

                5: { score: 0, complete: false }

            };

        }

A *resetScore()* metódus visszaállítja a szintekhez tartozó pontszámot 0-ra és a szint teljesítését hamisra, valamint törli a localStorage-ba elmentett adatokat a *localStorage.removeItem()* segítségével. A reset után meghívom a *draw()* függvényt, ami a játékosnak vizuálisan is megjeleníti a változásokat.

resetScore() {

    for (let level = 1; level <= 5; level++) {

        this.highscores[level].score = 0;

        this.highscores[level].complete = false;

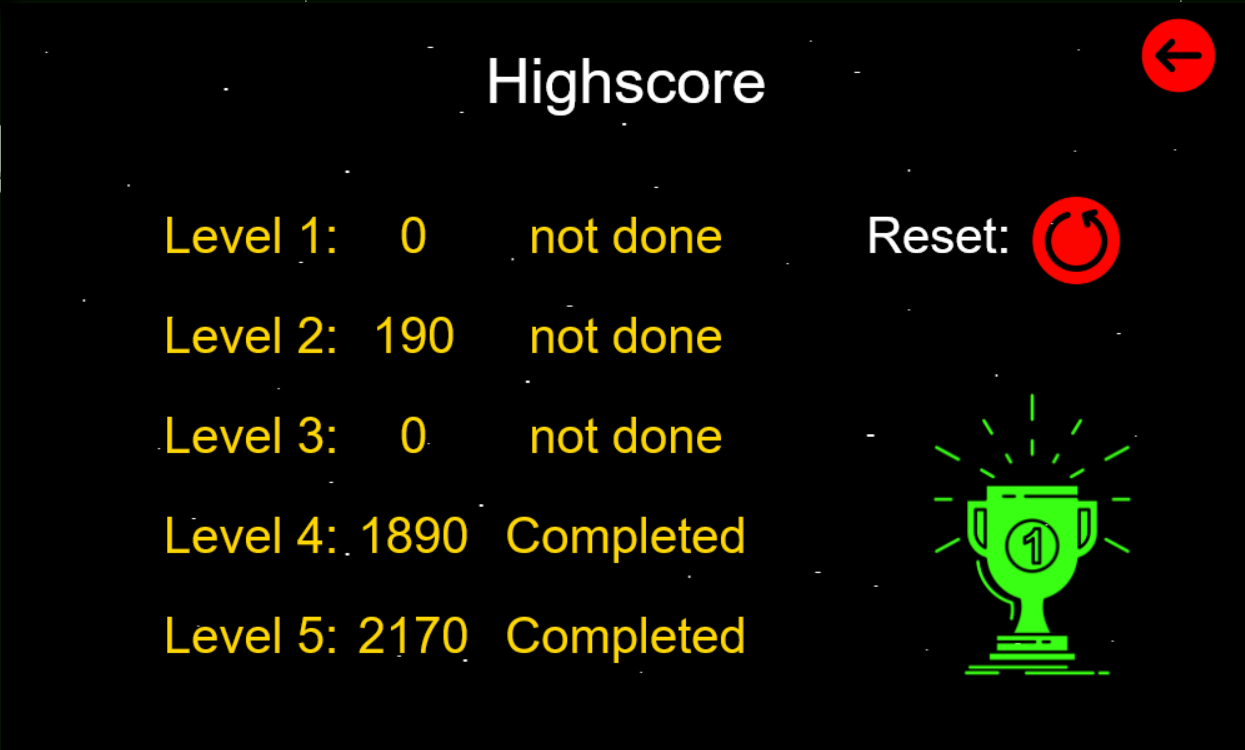
    }

    // highscores törlése a localStorage-ból

    localStorage.removeItem('highscores');

    this.draw();

}



14.ábra: Highscore menüpont kinézete

## 8.9 Játékbeli fizetőeszköz használata

Az in-game currency, jelen esetben a rubin gyűjtése kulcsfontosságú szerepet játszik a játékélmény javításában. Ezzel a lehetőséggel a játékosok vásárolhatnak űrhajókat, amikkel jobb eredményeket érhetnek el a játék során, mivel minden jármű egyedi képességekkel rendelkezik. Továbbá egyedülálló hátterek vásárlására is lehetőségük adódik a *costumization* menüpont megnyitásával. A rubin gyűjtése mindemellett motiválja a felhasználókat, hogy még több időt szánjanak a játékra, hiszen folyamatos jutalomérzetet ad. A játékos dönthet, hogy mire használja fel a fizetőeszközt, amely egyfajta bevonódást és döntési szabadságot ad a játékosnak.

A játékbeli fizetés 2 helyen került megvalósításra a kódban: az *index.js* és a *costumization.js* fájlokban. Itt 2 különböző funkció valósul meg: az *index.js* a rubin szerzéséért felel, ezzel szemben a *costumization.js* a rubin felhasználásával kapcsolatos tranzakciókat kezeli és számon tartja a felhasználó rubin mennyiségét.

A rubin növelése akkor történik meg, ha a játékos sikeresen teljesít egy szintet. A szinttel járó bónusz az alábbi logikával lett kiszámolva. Majd ezek után a rubin értékét növeljük és ezt az értéket frissítjük a *costumization* osztályban is:

const rubinBonus = level \* 25 + 25;

if (levelComplete) {

   increaseRubin(rubinBonus);

}

function increaseRubin(amount) {

    rubin += amount;

    costumization.updateRubin(rubin);

}

A *costumization* osztály, ahogy már említettem az in-game currency menedzseléséért felel. Többek közt itt kerülnek elköltésre a rubinok.

A vásárlások előtti ellenőrzésért a *canAfford()* függvény felel, amely azt vizsgálja, hogy van-e elegendő rubinja a játékosnak a tranzakció lebonyolításához.

canAfford(amount) {

    return this.rubin >= amount;

}

Az elköltött rubin levonása a *decreaseRubin()*, frissítése az *updateRubin()* függvénnyel történik:

decreaseRubin(amount) {

    this.rubin -= amount;

    this.updateRubin(this.rubin);

}

updateRubin(newRubin) {

    this.rubin = newRubin;

}

A tranzakció lebonyolítása kattintás hatására az eseménykezelő aktiválásával kezdődik, amely a vásárlási folyamatot indítja el. Először is azt visgáljuk, hogy a felhasználó a megfelelő gombra kattintott rá a koordináták behatárolása alapján, majd azt hogy rendelkezik-e már a megvásárolni kívánt elemmel. Ezután pedig azt ellenőrzöm, hogy rendelkezünk-e elegendő rubinnal a vásárláshoz. Ha ezek a feltételek teljesülnek, akkor jelen esetben megvásároltuk a player2 karaktert és levontunk érte 50 rubint, majd a képernyő újrarajzolásával vizuálisan is jelezzük a vásárló számára a tranzakció eredményét.

if(x >= 190 && x <= 240 && y >= 420 && y <= 440 && this.player2Bought === false) {

       if (this.canAfford(50)) {

           this.player2Bought = true;

           this.decreaseRubin(50);

           this.draw();

       }

}

# 9. Összefoglalás

A dolgozat célja a HTML5 canvas alapú játékfejlesztés alapos vizsgálata, amely az elméleti alapoktól egészen egy konkrét projekt: a Galactic Warrior nevű játék megvalósításáig terjed. Az első részben a játékfejlesztés iparágának átfogó elemzése található, amely tartalmazza a játékfejlesztés történelmi fejlődését, a mai piac működését, valamint a technológiai törekvéseket, amelyek meghatározzák az ágazat irányvonalait. A dolgozatban részletesen bemutatom azokat a legfontosabb technológiákat, amelyeket a fejlesztők használnak, különösen a HTML5 és a canvas jelentőségére koncentrálva. Ezek a technológiák segítsége lehetővé teszik a böngészőalapú játékok fejlesztését, ami napjainkban egyre nagyobb népszerűségnek örvend. Az elemzés kiterjed a különböző keretrendszerek, játékfejlesztési motorok és könyvtárak szerepére is, amelyek megkönnyítik és felgyorsítják a fejlesztők munkáját. A technológiai eszközök ismertetése után a dolgozat bemutatja a webes adattárolási lehetőségeket, különösen az elkészített játék során is felhasznált localStorage működését, amellyel a játékos adatai a böngészőben történő tárolása megoldható és visszakereshető. Erre azért van szükség, hogy megoldást nyújtson hosszútávon a játékos eredményeinek és egyéb adatainak letárolására. Az adatkezelési lehetőségek mellett a dolgozat érinti a digitális tartalmak és licencek jogi kérdéseit, amelyek lényeges tájékoztatást adnak a játékfejlesztők számára, különösen ha harmadik féltől származó tartalmakat használtak fel a fejlesztés során.

A dolgozat második felében az elkészített játék: a Galactic Warrior kerül részletes bemutatásra. A játék egy űr tematikájú, space invaders stílusú játék, amely a HTML5 canvas technológia felhasználásával készült el. Elsőként ismertetem a játék tervezési folyamatait, kezdve a játék témájának és alapszerkezetének ismertetésével, egészen a technológiai megvalósításokig. Itt kiemelt szerepet kapnak a HTML5 canvas által nyújtott lehetőségek, amely hozzájárult egyes grafikai elemek, karakterek és animációk kirajzolásához. A technológiai részletek bemutatása során kitér a dolgozat az eseménykezelők használatára, amelyek a felhasználói interakciók kezelésében játszanak kulcsszerepet, például billentyűzet vagy egér használat során. A dokumentum a játéklogika megvalósításával is foglalkozik, beleértve a karakterek mozgását, a támadási rendszert, a pontszerzést és a játékmenet nyomon követését. A felhasználói élmény növelése is nagy hangsúlyt kapott a játék elkészítése folyamán a játékbeli fizetőeszköz bevezetésével, amellyel lehet venni pályákat és űrhajókat. Ez mellé a highscore funkció is kidolgozásra került a játékos legjobb eredményeinek elmentésének érdekében.

Mindent összevetve a dokumentum átfogó képet nyújt a HTML5 canvas alapú játékfejlesztés gyakorlati és elméleti áttekintéséről. Az elméleti háttér bemutatása után a konkrét játékfejlesztési projekt elemzése kiemeli a fejlesztési folyamat során felmerülő technikai és kreativitásbeli kihívásokat, valamint azok megoldásait. A dolgozat célja bemutatni egy HTML5 canvas technológiával készített interaktív és vizuálisan vonzó játékot, amely a böngészőben futtatható, miközben a webes technológiák által kínált modern megoldásokra támaszkodik.

# 10. Summary

The aim of the thesis is to deep examine HTML5 canvas-based game development, which extends from the theoretical foundations to the implementation of a specific project: the game called Galactic Warrior. The first part provides a comprehensive analysis of the game development industry, including the historical development of game development, how the market operates today, and the technological aspirations that shape the direction of the industry. In the thesis, I present in detail the most important technologies used by developers, focusing especially on the importance of HTML5 and canvas. These technologies enable the development of browser-based games, which are increasingly popular these days. The analysis also covers the role of different frameworks, game development engines and libraries that facilitate and speed up the work of developers. After the description of the technological tools, the thesis presents the web data storage options, especially the operation of localStorage, which is also used during the created game, with which the player's data can be stored in the browser and retrieved. This is necessary in order to provide a long-term solution for storing the player's results and other data. In addition to data management options, the thesis touches on the legal issues of digital content and licenses, which provide essential information for game developers, especially if third-party content was used during development.

In the second half of the thesis, the completed game: Galactic Warrior is presented in detail. The game is a space themed, space invaders style game, which was created using HTML5 canvas technology. First, I will describe the game's design processes, starting with the theme and basic structure of the game, all the way to the technological implementations. Here, the possibilities provided by the HTML5 canvas, which contributed to the drawing of some graphic elements, characters and animations, play a prominent role. During the presentation of the technological details, the thesis covers the use of event handlers, which play a key role in managing user interactions, for example when using a keyboard or mouse. The document also covers the implementation of game logic, including character movement, attack system, scoring, and gameplay tracking. Enhancing the user experience was also given a lot of emphasis during the creation of the game with the introduction of in-game currency, which can be used to buy tracks and spaceships. In addition, the highscore function was developed in order to save the best results of the player.

All in all, the document provides a comprehensive overview of the practical and theoretical overview of HTML5 canvas-based game development. After presenting the theoretical background, the analysis of the specific game development project highlights the technical and creative challenges that arise during the development process, as well as their solutions. The aim of the thesis is to present an interactive and visually attractive game made with HTML5 canvas technology, which can be run in the browser while relying on modern solutions offered by web technologies.

# 11. Irodalomjegyzék

*[1]* [*https://hu.wikipedia.org/wiki/Videójáték*](https://hu.wikipedia.org/wiki/Videójáték)

*[2]* [*https://hu.wikipedia.org/wiki/Játékszer*](https://hu.wikipedia.org/wiki/Játékszer)

*[3]* [*https://www.grandviewresearch.com/industry-analysis/video-game-market*](https://www.grandviewresearch.com/industry-analysis/video-game-market)

*[4]* [*https://www.futuremarketinsights.com/reports/video-game-market*](https://www.futuremarketinsights.com/reports/video-game-market)

*[5]* [*https://www.voronoiapp.com/entertainment/50-Years-of-Video-Game-Industry-Revenues-by-Platform-456*](https://www.voronoiapp.com/entertainment/50-Years-of-Video-Game-Industry-Revenues-by-Platform-456)

*[6]* [*https://newzoo.com/resources/blog/last-looks-the-global-games-market-in-2023*](https://newzoo.com/resources/blog/last-looks-the-global-games-market-in-2023)

*[7]* [*https://drawandcode.com/learning-zone/what-is-a-game-engine*](https://drawandcode.com/learning-zone/what-is-a-game-engine)

*[8]* [*https://www.devopsschool.com/blog/game-development-frameworks-a-comprehensive-guide/*](https://www.devopsschool.com/blog/game-development-frameworks-a-comprehensive-guide/)

*[9]* [*https://spotlighthungary.hu/2024/01/legnepszerubb-jatek-keszito-eszkozok-es-platformok/*](https://spotlighthungary.hu/2024/01/legnepszerubb-jatek-keszito-eszkozok-es-platformok/)

*[10]* [*https://dotnet.microsoft.com/en-us/apps/games/unity*](https://dotnet.microsoft.com/en-us/apps/games/unity)

*[11]* [*https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-programming-and-scripting*](https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-programming-and-scripting)

*[12] https://en.wikipedia.org/wiki/GameMaker*

*[13]* [*https://blog.logrocket.com/best-javascript-html5-game-engines/*](https://blog.logrocket.com/best-javascript-html5-game-engines/)

*[14] https://medium.com/gamezop/%EF%B8%8F-what-are-the-best-html5-game-engines-80646633913e*

*[15]* [*https://medium.com/stackanatomy/svg-vs-canvas-a-comparison-1b58e6c84326*](https://medium.com/stackanatomy/svg-vs-canvas-a-comparison-1b58e6c84326)

*[16]* [*https://developer.mozilla.org/en-US/docs/Web/SVG*](https://developer.mozilla.org/en-US/docs/Web/SVG)

*[17]* [*https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage*](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage)

*[18]* [*https://www.loginradius.com/blog/engineering/guest-post/local-storage-vs-session-storage-vs-cookies/*](https://www.loginradius.com/blog/engineering/guest-post/local-storage-vs-session-storage-vs-cookies/)

*[19]* [*https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB\_API/Basic\_Terminology*](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Terminology)

*[20]* [*https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API*](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)

*[21]* [*https://creativecommons.org/share-your-work/cclicenses/*](https://creativecommons.org/share-your-work/cclicenses/)

*[22]* [*https://opengameart.org/content/faq#q-proprietary*](https://opengameart.org/content/faq#q-proprietary)

*[23]* [*https://en.wikipedia.org/wiki/Space\_Invaders*](https://en.wikipedia.org/wiki/Space_Invaders)

Az irodalomjegyzékben felsorolt linkek utolsó elérési dátuma: 2024. 10. 28

# 12. Mellékletek:

**Teljes kód (digitális melléklet)**

A dolgozathoz mellékelve megtalálható egy **CD / pendrive**, amelyen a program teljes forráskódja a Galactic warrior mappában elérhető. A digitális melléklet a következő fájlokat tartalmazza:

* index.html – A fő HTML fájl, amely a játék alapstruktúráját biztosítja
* stlye.css - A játék vizuális megjelenéséért felelős stíluslap
* index.js – A fő programfájl, amely koordinálja a játék működését

**Ellenségek kezelése**

Az egyes szinteken szereplő ellenségek irányításáért felelős fájlok:

* EnemyHandler1.js, EnemyHandler2.js, EnemyHandler3.js, EnemyHandler4.js, EnemyHandler5.js

**Egyéb menüpontok**

Menüpontokat kezelő fájlok:

* HighScore.js – Legmagasabb elért pontszámok mentése és megjelenítése Description.js – Játékismertető, amely egy rövid kedvcsinálót tartalmaz
* Sound.js – A játék hangjának, hangerőnek kezelése
* Costumization.js - Testreszabási lehetőségek és vásárlási opciók

**Grafikai elemek és mozgások megvalósítása**

Az egyes játékobjektumok megjelenítéséért és mozgatásáért felelős fájlok:

* Enemy.js, Bullet.js, Player.js, PlayerBullet.js, MiniBoss.js, Boss.js, BossBullet.js, Gift.js, Gift2.js, Meteor.js, MeteorStand.js

**Grafikai elemek kezelése**

Különböző grafikai elemek irányításához szükséges fájlok:

* BulletController.js, GiftController.js, Gift2Controller.js, MeteorController.js

**Mozgásvezérlés**

* MovingDirection.js – ellenfelek mozgását vezérli

**Médiatartalmak**

A következő mappák tartalmazzák a játékban felhasznált képeket és hangokat:

* images: A játékban felhasznált képek, ideértve a karakterek, háttérképek és más vizuális elemek képfájljait
* sounds: a játék hangeffektusai és háttérzenéi

**Futtatási\_Útmutató.txt:**  
A játék futtatásához szükséges információkat tartalmazó fájl. Ez a dokumentum részletes útmutatást nyújt a projekt megnyitásáról és futtatásáról.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Mileff Péternek, aki nélkül a szakdolgozatom nem jöhetett volna létre. Őszinte hálával tartozom az elméleti és gyakorlati útmutatásaiért, valamint erkölcsi támogatásáért, amellyel a kutatás több szakaszában is segített. Az általa adott szakmai tanácsok és a személyes biztatás hozzájárult a választott téma mélyebb megértéséhez és, hogy eredményesen megvalósítsam kitűzött céljaimat. Az ő segítsége jelentős mértékben hozzájárult ahhoz, hogy a projekt elérje mostani formáját és színvonalát.