**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Informatikatudományi Intézet**

**Programozási Nyelvek és Fordítóprogramok Tanszék**

**Kétdimenziós platformjáték C# és Unity alapon**

Szerző: Témavezető:

**Fodor Attila Krisztián** **Pataki Norbert**

Programtervező informatikus BSc. Habilitált egyetemi docens

**Budapest, 2025**

Tartalomjegyzék

[Köszönetnyilvánítás 1](#_Toc196842446)

[1. Bevezetés 2](#_Toc196842447)

[1.1. Motiváció 2](#_Toc196842448)

[1.2. Szakdolgozatom témája 2](#_Toc196842449)

[1.3. Készítés során felmerülő főbb akadályok 3](#_Toc196842450)

[2. Felhasználói dokumentáció 4](#_Toc196842451)

[2.1. Rendszerkövetelmények 4](#_Toc196842452)

[2.2. A játék telepítése és futtatása 4](#_Toc196842453)

[2.3. A játék ismertetése 5](#_Toc196842454)

[2.3.1. A játékmenet leírása 5](#_Toc196842455)

[2.3.2. Főmenü 5](#_Toc196842456)

[2.3.3. Beállítások 6](#_Toc196842457)

[2.3.4. Új játék 6](#_Toc196842458)

[2.3.5. Játékfelület 7](#_Toc196842459)

[2.3.6. Szüneteltetés menü 10](#_Toc196842460)

[2.3.7. Irányítás 11](#_Toc196842461)

[2.3.8. Pályaelemek 14](#_Toc196842462)

[2.3.9. Csontvázak 18](#_Toc196842463)

[2.3.10. Pályák 20](#_Toc196842464)

[2.3.11. Győzelmi képernyő 22](#_Toc196842465)

[3. Fejlesztői dokumentáció 24](#_Toc196842466)

[3.1. Használt eszközök és technológiák 24](#_Toc196842467)

[3.1.1. Unity Engine választásának indoklása 24](#_Toc196842468)

[3.1.2. C# programozási nyelv választásának indoklása 25](#_Toc196842469)

[3.1.3. A dolgozathoz használt fontos Unity fogalmak 25](#_Toc196842470)

[3.2. Megoldási terv 31](#_Toc196842471)

[3.3. Felhasználói esetek 31](#_Toc196842472)

[3.4. A játék fontosabb scriptjei (megvalósítási terv) 31](#_Toc196842473)

[3.5. A scriptek kapcsolatai (megvalósítási terv) 31](#_Toc196842474)

[3.6. A játék tesztelése 31](#_Toc196842475)

[3.6.1. Manuális tesztelés 31](#_Toc196842476)

[3.6.2. Automatizált tesztelés Unit Testekkel 32](#_Toc196842477)

[4. Összefoglalás 34](#_Toc196842478)

[5. Fejlesztési lehetőségek 35](#_Toc196842479)

[6. Melléklet és felhasznált források 37](#_Toc196842480)

[6.1. Felhasznált Unity csomagok 37](#_Toc196842481)

[6.2. Grafikus anyagok 38](#_Toc196842482)

[6.3. Hang és zenei anyagok 39](#_Toc196842483)

[6.4. Egyéb források 39](#_Toc196842484)

# Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindazoknak, akik hozzájárultak a *Dungeon Crawler* játék jelen verziójának megvalósításához.

Először is köszönettel tartozom a témavezetőmnek, Pataki Norbertnek, hogy betöltötte ezt a szerepet. A rendszeresen tartott konzultációink, a gyors és pontos visszajelzései, rugalmassága nélkül ez a dolgozat nem készült volna el.

Hálás vagyok édesapámnak, aki az egész fejlesztés alatt folyamatos támogatást nyújtott. Bár nincs otthon a programozás világában, mégis ösztönzött, noszogatott, biztatott. Érdeklődött és meghallgatott, amikor kétségbeesetten próbáltam elmagyarázni egy újabb hiba technikai oldalát és lehetséges forrásait. Megpróbálta megérteni, amit felvázolok, még akkor is, amikor a nagy része teljesen absztrakt és megfoghatatlan volt számára. A vele folytatott beszélgetések sokszor vezettek felfedezésekhez és megoldásokhoz, melyekre csak oly módon tehettem szert, hogy ténylegesen kimondtam, elmagyaráztam egy-egy felmerülő problémát.

Továbbá szeretném megköszönni a barátnőmnek, aki végtelen türelmével és biztatásával még akkor is támaszt nyújtott, amikor ő magának is szüksége lett volna rá.

Szeretném még köszönetemet kifejezni édesanyámnak és a nagyszüleimnek, akikre mindig számíthattam, amikor a dolgozat megszületésében kételkedtem. Szavaik hatására minden alkalommal megújult erővel folytattam a dolgozat írását.

Végül, de nem utolsó sorban szeretném megköszönni mind az egyetemen megismert, mind a régebbről megmaradó barátaimnak, akik nélkül elképzelhetetlennek tartom a szakdolgozatom megszületését. A tőlük kapott támogatás, odaadás és visszajelzés tette lehetővé, hogy a legnehezebb kihívásokkal is szembenézzek.

# Bevezetés

## Motiváció

A videójátékok mindig is fontos szerepet töltöttek be az életemben. Már gyerekkoromban is kikapcsolódást, menedéket jelentettek - egy módot arra, hogy eltereljem a figyelmemet a mindennapi problémákról. Bár ma már leginkább kompetitív, valós idejű játékokat kedvelek, a kezdetek platformjátékokhoz kötődnek: az első játékom egy oldalnézetes platformjáték volt, amit egy Playstation 2 konzolon játszottam. Akkor természetesen még nem foglalkoztatott, hogyan működik egy játékkonzol. Nem kérdőjeleztem meg, mi történik a „fedél alatt”, azt, hogy mi minden szükséges ahhoz, hogy a kontroller bizonyos gombnyomásaira azonnali vizuális visszajelzést kapjak a tévémtől. Nem gondolkodtam azon sem, hogy mennyi munka van egy ilyen játékélmény megteremtése mögött. Csupán élveztem, és megfeledkeztem minden másról - mintegy elvarázsolva. Ahogyan azonban telt az idő, egyre jobban érdekelni kezdett, hogy hogyan működnek ezek a világok a háttérben: hogyan áll össze egy játék logikája, hogyan lesz a sok apró darabból egy teljes, koherens kép. Ezért is választottam a témámat a játékfejlesztés világából.

## Szakdolgozatom témája

A szakdolgozatom célja egy kétdimenziós platformjáték fejlesztése, amelyben a játékos feladata a kiszabadulás. A platformjáték, vagy más néven „platformer*”*, egy olyan videójáték-műfaj, amelyben a játékos karakterének mozgása, különösen az ugrás és más vertikális mozgásformák, kiemelt szerepet kap. A műfaj jellegzetessége, hogy a játékos különböző szinteken, emelvényeken, akadályokon, tereptárgyakon - tehát platformokon - navigálva halad előre, gyakran időzítésre és ügyességre épülő kihívásokkal szembesülve. A platformjátékok gyakran kombinálják az ügyességi elemeket harci, felfedező vagy logikai mechanikákkal, ezáltal változatos játékmenetet kínálva. A dolgozatom témájaként szolgáló játék tehát ebbe a műfajba illeszkedik. A kiszabaduláshoz a játékosnak különböző pályákat kell felfedeznie, olyan mozgási mechanikákat felhasználva, mint például a futás, ugrás, illetve dupla ugrás, hiszen bizonyos pályaelemeket csupán ilyen módon képes elkerülni. A pályákon különféle ellenfelek találhatók, amelyek mind eltérő módon próbálják megállítani a játékost a kiszabadulásban. A felhasználónak a játék során erősítések (*powerupok*) gyűjtésére is lehetősége van. Ezek keresése és használata nem kötelező, de erősen ajánlott, mivel ezek teszik lehetővé bizonyos pályaszekciók elérését, és az ellenfelekkel való küzdelemben is nagy segítséget nyújthatnak.

## Készítés során felmerülő főbb akadályok

Ez a projekt az első egyedülálló játékfejlesztési projektem, és számos kihívás elé állított. A Unity játékmotort korábban nem használtam, így a kezdeti lépések során több alapvető problémával szembesültem. Először is akadályt jelentett a motor felépítésének, működésének és logikájának megértése. A különböző scriptelési lehetőségek, valamint a játékmenet működését vezérlő eseményrendszer mind kidolgozatlan fogalmak voltak a fejemben, ezért jelentős időt kellett szánnom a megértésükre. Eleinte problémát jelentett a jelenetek közötti átmenetek és azok játékelemekre való hatásának megértése is, de számos más Unity-specifikus működést meg kellett tanulnom. Akadály volt még például az animációk készítése is, pontosabban az animációk közötti átmenetek feltételeinek finomhangolása és kezelése, amely mind játékmotoron, mind kódon belül megfontolt konfigurációt igényelt. A fejlesztés során előfordult, hogy egyes dolgokat a kezdeti, hiányos ismereteimből kiindulva nem megfelelő módon valósítottam meg, ami miatt később problémákba ütköztem. A dokumentációk, fórumok és oktatóanyagok használata ekkor elengedhetetlenné vált, főleg, amikor egy-egy ilyen hiba okát nehéz volt pontosan beazonosítani. Ám amikor a felmerülő problémát megvizsgáltam, komponensekre bontottam, majd a rendelkezésre álló segédanyagok segítségével eljutottam egy helyes megoldásra, óriási sikerélményben volt részem. Technikai nehézséget okozott viszont, hogy a félév nagy részében csak otthoni asztali gépen tudtam dolgozni, mivel a jelenlegi, fejlesztéshez alkalmas laptopomat csak a szemeszter végére sikerült beszereznem. Emiatt sokszor voltam a számítógéphez kötve, ami nemcsak fizikailag volt megterhelő, hanem a társas kapcsolataimra is hatással volt.

# Felhasználói dokumentáció

## Rendszerkövetelmények

A dolgozat elkészítéséhez a Unity 6000.0.37f1-es verzióját használtam, aminek az alábbiak a minimum rendszerkövetelményei:

|  |  |
| --- | --- |
| **Operációs rendszer** | Windows 10 21H1-es verzió (19043-as build) vagy újabb (64 bites), illetve Windows 11 21H2-es verzió (22000-as build) vagy újabb (Arm64) |
| **Processzor** | X64 architektúra SSE2 instrukciós szett támogatással, Arm64 |
| **Grafikus API** | DX10-, DX11-, DX12- vagy Vulkan-kompatibilis grafikus kártya |
| **További követelmények** | A hardvergyártó által hivatalosan támogatott illesztőprogramok szükségesek. |

## A játék telepítése és futtatása

A program használatához - a letöltött tömörített állomány kicsomagolása után - a „Build/DungeonCrawler.exe” futtatható állományt kell futtatni. Ezenkívül explicit telepítésre nincsen szükség.

## A játék ismertetése

### A játékmenet leírása

A játék egy oldalnézetes, kétdimenziós platformer, amely egy elfeledett föld alatti tömlöcben veszi kezdetét. Főhősünk a csontvázharcosok fogságába esett, és most a játékos segítségével próbál kiszabadulni. A cél minden pályán megtalálni a kijárathoz vezető kulcsot, leküzdeni az útba eső akadályokat és ellenfeleket, valamint felhasználni a különféle erősítéseket a továbbjutáshoz. A játék támogatja a mentést és visszatöltést is. Betöltéskor a játékos a legutóbb aktivált checkpointnál folytatja a kalandot, és az állapotok - például az életpontok, a próbálkozások száma, az összegyűjtött erősítések, valamint a megszerzett pontszám és legyőzött ellenfelek száma - mind megőrzésre kerülnek. Az interaktív pályaelemek, például a széttört objektumok vagy felvett tárgyak, az aktuális állapotuknak megfelelően jelennek meg újratöltés után is.

### Főmenü

A program futtatásakor megjelenő főmenüben (1. ábra: Főmenü) a játék neve alatt négy gomb jelenik meg. Ezek közül a legfelső *New Game* gomb lehetőséget kiválasztva a felhasználó új játékot indíthat a bevezető pályáról kezdve. A *Continue* gombra kattintva a játékos folytatja az elmentett játékmenetet a legutoljára aktivált checkpointnál. Az *Options* gomb megnyomása szintén az elvárt hatást éri el, átirányít a beállítások menüjébe. A *Quit* gomb használatával pedig a játékos bezárhatja a programot.



1. ábra: Főmenü

### Beállítások

A beállítások menüben (2. ábra: Beállítások) az *Options* felirat alatt három csúszka található. A legfelső, *Main* csúszka a játék általános hangerejét szabályozza, a második, *Music* csúszkával a háttérzene hangereje módosítható, míg a harmadik, *SFX* csúszkával a játékmenetben használt hangeffektek erőssége állítható be. A felhasználónak lehetősége van továbbá a kívánt felbontás és a teljes képernyős nézet konfigurálására. Amikor pedig a főmenüre szeretne visszatérni, a képernyő alján megtalálható *Back* gombot kell használnia. A beállítások indítások között is megmaradnak.



2. ábra: Beállítások

### Új játék

Eleinte mind a *New Game*, mind a *Continue* gomb hatására a *Tutorial* pályára kerülünk. Ez a rabul ejtésünkre szolgáló kazamatában található. Itt a pályán felfüggesztett szöveges utasítások segítségével a játékos megkezdheti a játékkal és az irányítással való ismerkedést. Először viszont kitérnék a játék felületére és működésére.

### Játékfelület

A játék felhasználói felülete egyszerű, letisztult és intuitív módon tájékoztatja a játékost, ikonokat és egyszerű szövegeket alkalmaz.

#### Életerő-sáv

A képernyő bal felső sarkában látható az életerő-sáv (3. ábra: Életerő-sáv), amely százalékos értékű, alatta pedig egy felirat jelzi a karakter aktuális életpontjait.



3. ábra: Életerő-sáv

A felirat alatt további, a játékállapot szempontjából elengedhetetlen információ-tartalommal rendelkező felületi elemek találhatók.

#### Életek

A hátralévő próbálkozások (4. ábra: Életek) számát három darab szív ikon jelzi, vagy betöltött, vagy betöltetlen állapotban. Ezek az ikonok jól láthatóak, így a játékos mindig pontosan tudja, mennyi hibalehetősége maradt.



4. ábra: Életek

#### Erősítéstár

A közvetlenül a szívektől jobbra látható erősítéstár (5. ábra: Erősítéstár) mutatja, hogy a játékosnak milyen erősítései milyen mennyiségben vannak eltárolva. A három különböző erősítésfajtának pontos működésére később kitérek, egyelőre viszont azt szeretném kiemelni, hogy bizonyos fajtákból létezik több alfajta (például: kis életerő/nagy életerő erősítés). Ezen alfajták játékosra gyakorolt hatása ugyanaz, csupán a hatás mértéke vagy időtartama változik.

A képen képernyőkép, rajzfilm, művészet látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

5. ábra: Erősítéstár

Az egyes alfajták más megjelenésűek, és bár hasonló ikonjuk van, mégis fontos megkülönböztetni őket. Ennek oka az, hogy amelyiknek az ikonját az erősítéstár éppen mutatja, az az alfajta lesz - a megfelelő gyorsgomb megnyomásakor - elhasználva. Az elhasználás azt az erősítést részesíti előnyben, amelyet a játékos hamarabb megszerzett, nem pedig amit legutóbb. Az erősítés típusok megjelenési sorrendje nem meghatározott, a felvétel sorrendjében jelennek meg a szívek után, balról jobbra sorakozva. Az egyes fajták ikonjain elhelyezkedő szám jelzi, hogy mennyi maradt még abból a típusból a játékos tárában.

#### Tűzlabda indikátor

Az elhasználható erősítések után helyezkedik el a tűzlabda erősítés indikátora (6. ábra: Tűzlabda indikátor), mely felvételre megjelenik, következő szintig pedig láthatóvá válik.

A képen pixel, képernyőkép, Színesség, tervezés látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

6. ábra: Tűzlabda indikátor

#### Kulcs indikátor

A kulcs indikátor (7. ábra: Kulcs indikátor) hasonló módon működik, ha egyszer azt a játékos megszerezte, a pálya erejéig aktívvá válik.

A képen képernyőkép, pixel látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

7. ábra: Kulcs indikátor

#### Pontszámláló

A játékban pontokat kell gyűjteni. Ezen pontok megjelenítéséhez mentések között perzisztens számlálók találhatók meg a képernyőn. A fent felsorolt játékfelületi elemek alatt, a képernyő bal oldalán jelenik meg a pontszámláló (8. ábra: Pontszámláló), mely a játékos által összegyűjtött összpontszámot mutatja. Ez minden legyőzött ellenfél vagy elpusztított pályaelem után frissül.



8. ábra: Pontszámláló

#### Csontváz számláló

A képernyő jobb felső sarkában fellelhető a csontváz számláló, amely minden legyőzött csontváz után növeli értékét. A számláló növekedésekor az összpontszám is változik. 100 pont jár egy kardforgató vagy pörölyös, 200 pedig egy távolsági csontvázharcos legyőzéséért. A 9. ábrán láthatjuk, hogy a felhasználó az új játék kezdete óta összesen 1 darab csontvázat tett el láb alól.



9. ábra: Csontváz számláló

#### Törés számláló

A 10. ábrán látható számláló közvetlenül a fenti alatt jelenik meg, minden széttört pályaelem után eggyel nő az értéke. Ez szintén felülírja az összesített pontszámot, elemenként 50 pontot jelent.



10. ábra: Törés számláló

A HUD (*head-up display*) minden eleme úgy van elhelyezve, hogy az ne zavarja meg a játékélményt, ugyanakkor minden fontos információ könnyen elérhető és gyorsan értelmezhető legyen a játékos számára. A dizájn célja az volt, hogy a felhasználó számára ne okozzon fennakadást a pályák felfedezése és az akció közbeni döntéshozatal.

### Szüneteltetés menü

A játék szüneteltetése elengedhetetlen része bármely játéknak, hiszen valamilyen módon „kiutat” kell biztosítani a felhasználónak - akár a főmenübe, akár az alkalmazás bezárásához. Ezt a funkciót tölti be a 11. ábrán látható menüpont, amelyet a felhasználó a játék közben az *Escape* billentyű lenyomásával érhet el. A megjelenő menü négy gombot tartalmaz: a játékmenet folytatását lehetővé tevő *Resume* gombot, a főmenüben már bemutatott beállításokat megnyitó *Options* gomb, a kezdőképernyőre visszalépést biztosító *Back to Menu* gombot, valamint a programot bezáró *Quit* gombot. A játék bármely módon történő elhagyásakor automatikus mentés történik.

A képen szöveg, képernyőkép, Multimédiás szoftver, áramkör látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

11. ábra: Szüneteltetés menü

### Irányítás

#### Mozgás

A játékos a karaktert az **A** vagy a **Balra nyíl** billentyű lenyomásával balra, illetve a **D** vagy a **Jobbra nyíl** billentyű lenyomásával jobbra tudja irányítani (12. ábra: Mozgás). A **Bal Shift** vagy **Jobb Shift** billentyű egyidejű lenyomva tartásával a karakter futni kezd az adott irányba.

A képen képernyőkép, szöveg látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

12. ábra: Mozgás

A **W**, **Felfelé nyíl** vagy **Space** billentyűk lenyomásával a játékos ugrást (13. ábra: Ugrás) hajthat végre. Amennyiben a karakter a levegőben tartózkodik, és ismételten lenyomja az ugrásra szolgáló billentyűk egyikét, egy második ugrást (dupla ugrást) hajt végre. Fontos megjegyezni, hogy platformról leesés után is csak egyetlen extra ugrás áll rendelkezésre.

A képen képernyőkép, szöveg, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

13. ábra: Ugrás

#### Küzdelem

A közelharci támadás (14. ábra: Közelharc) a **Bal egérgomb** vagy az **O** billentyű lenyomásával aktiválható, amikor a játékos a földön áll. Az első ütés 10 sebzést okoz és kisebb visszalökődést (*knockback*-et) eredményez az ellenségen. Egy rövid időablakon belül végrehajtott második ütés 20 sebzést okoz, és valamivel nagyobb visszalökődést vált ki.

A képen képernyőkép, Videojáték-szoftver, Számítógépes játék, Digitális képszerkesztés látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

14. ábra: Közelharc

A távolharci támadások (15. ábra: Távolharc) a **Jobb egérgomb** vagy a **P** billentyű lenyomásával indíthatók. Ha a játékos nem rendelkezik tűzlabdával, követ hajít el az aktuális nézési irányba, amely 10 sebzést okoz és kisebb visszalökődést eredményez. Amennyiben a játékosnál tűzlabda található, azt lövi ki, amely 20 sebzéssel és nagyobb visszalökő hatással jár. Mindkét támadást csak földön állva, az animáció ideje alatt helyben maradva lehet végrehajtani.

A képen képernyőkép, Digitális képszerkesztés, Videojáték-szoftver, Számítógépes játék látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

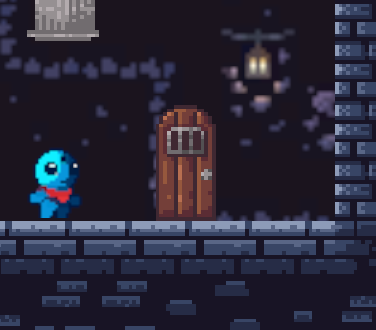
15. ábra: Távolharc

#### Erősítés használata

A játékos különböző eltárolt erősítéseket alkalmazhat az **1**, **2**, illetve **3** számgombok segítségével. Az **1-es** gombbal életerőt növelő (*HealthBuff*) erősítést aktiválhat, feltéve, hogy nem teljes az életereje. A **2-es** gombbal sebességnövelő (*SpeedBuff*), míg a **3-as** gombbal gravitációcsökkentő (*GravityBuff*) hatást idézhet elő, amely magasabb ugrásokat tesz lehetővé. Fontos figyelmeztetés, hogy az újonnan használt erősítés az előző azonos típusú erősítés hatását azonnal felülírja, így a felelőtlen használat az erősítések gyors elpazarlásához vezethet. Figyelni kell még arra is, hogy a fenti számgombok mindig a megadott erősítéstípusból használják fel a soron következőt, függetlenül az erősítéstípusok *user interface*-en megjelenő sorrendjétől.

#### Pálya elhagyása

Amennyiben a játékos megszerezte a pályán elrejtett kulcsot, a kijárat közelében (16. ábra: Kijárat) az **E** billentyű megnyomásával átléphet a következő szintre.



16. ábra: Kijárat

Ekkor a képernyő lassan elsötétedik, amikor pedig az új pálya betöltődött, fokozatosan újra láthatóvá válik. A pályaelhagyás megtartja a játékos minden adatát, de a tűzlabda erősítést és az előzőleg megtalált kulcsot elveszíti, hiszen ezek az előző szint végéig érvényesek elemek.

### Pályaelemek

#### Platformok

A játék - műfajának nevéből adódóan - platformokon játszódik. A platformok a pályák felépítésének és a játékmenet dinamikájának alapvető építőkövei. Különböző típusú felületek lehetnek: egybefüggő szilárd pályaelemek, lebegő blokkok vagy kisebb-nagyobb emelvények. A játékos ezekre a platformokra képes felugrani, rajtuk mozogni, illetve róluk tovább ugrani. Ezeken fognak elhelyezkedni a széttörhető tárgyak és járőrözni az ellenséges csontvázak.

#### Éledési pontok

Minden pályán megtalálhatók jellegzetes kinézetű éledési pontok (17. ábra: Inaktív éledési pont), amelyek azt a célt szolgálják, hogy a játékos életvesztés esetén ne legyen kénytelen a szintet teljes egészében újrakezdeni, vagy ismételten végighaladni a már teljesített pályaszakaszokon. Ezek a pontok megjelenésükben egyértelműen eltérnek a környezettől, így könnyen felismerhetők. Minden éledési ponthoz tartozik egy prioritási érték: minél magasabb ez az érték, annál közelebb helyezkedik el a pálya kijáratához.



17. ábra: Inaktív éledési pont

Amikor a játékos először megközelít egy éledési pontot, az aktiválódik (18. ábra: Aktív éledési pont), arany színnel jelezve, hogy ettől kezdve a játékos ezen a ponton éled újra életvesztés után. Amennyiben a játékos egy már aktivált, vagy a jelenleginél alacsonyabb prioritással rendelkező checkpoint közelébe érkezik, nem történik változás, az aktuális éledési pont marad érvényben.



18. ábra: Aktív éledési pont

#### Csapdák

A pályákon elszórtan a platformokra telepített csapdák (19. ábra: Tüskék) találhatók. A játékosnak mindenképpen el kell kerülnie ezeket az akadályokat, ugyanis érintkezéskor a játékos - függetlenül aktuális életerőpontjaitól - azonnal elveszít egy életet.

A képen képernyőkép, Videojáték-szoftver, Digitális képszerkesztés, 3D modellezés látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

19. ábra: Tüskék

#### Ideiglenes erősítések

A játékban jelenleg három különböző ideiglenes erősítéstípussal találkozhatunk.

1. **Életerő erősítések (*HealthBuff*):**

* **Kicsi (Small):**
* legfeljebb 25 életerőpontot tölt vissza

20. ábra: Kis életerő

A képen rajzfilm, clipart, illusztráció látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Nagy (Large):**
* legfeljebb 50 életerőpontot tölt vissza

21. ábra: Nagy életerő

1. **Gyorsaság erősítések (SpeedBuff):**

* A képen labda látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**Kicsi (Small):**
* Sebességnövelés mértéke: 1.2-szeres
* Időtartam: 10mp
* Elhasználáskor felülírja az aktuális gyorsaság erősítés időtartamát és mértékét

22. ábra: Kis sebességnövelő

* A képen művészet látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**Nagy (Large)**
* Sebességnövelés mértéke: 1.4-szeres
* Időtartam: 5mp
* Elhasználáskor felülírja az aktuális gyorsaság erősítés időtartamát és mértékét

23. ábra: Nagy sebességnövelő

1. A képen rajzfilm, művészet látható

   Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**Gravitációs erősítés (GravityBuff):**

* Csökkenti a gravitációs mező erejét, magasabb ugrást és lassabb zuhanást eredményezve
* Időtartam: 6 mp

24. ábra: Gravitációcsökkentő

* Elhasználáskor felülírja az aktuális gravitációs erősítés időtartamát

#### Széttörhető pályaelemek

A széttörhető pályaelemeket a játékos támadásai mellett a lövedékek is elpusztíthatják. Bármilyen sebzés esetén azonnal összetörnek, megsemmisülésük után pedig 50% eséllyel tartalmazhatják a rájuk specifikus erősítést. A *powerup*-okhoz hasonlóan több típusú pályaelem különböztethető meg, mindegyik más gyakorisággal. A tárgy gyakorisága a benne rejlő erősítés mérlegelt hasznától függ. Például egy kis sebességgyorsítás kevesebb játékhelyzetben lehet olyan hasznos, mint 50 garantált életerő.

1. *A képen képernyőkép, pixel, tér látható

   Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.***Doboz (Box)**

* Kis életerő
* Kis sebességnövelő
* Gyakoriság: Gyakori (*Common*)

25. ábra: Doboz

1. A képen képernyőkép, pixel látható

   Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**Szürke váza (Grey vase)**

* Nagy sebességnövelő
* Gyakoriság: Nem gyakori (*Uncommon*)

26. ábra: Szürke váza

1. A képen képernyőkép, pixel, tervezés látható

   Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**Barna váza (Brown vase)**

* Gravitációcsökkentő
* Gyakoriság: Ritka (*Rare*)

27. ábra: Barna váza

1. A képen képernyőkép, pixel, tér látható

   Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**Hordó (Barrel)**

* Nagy életerő
* Gyakoriság: (*Epic*)

28. ábra: Hordó

#### A képen pixel, képernyőkép, Színesség látható Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.Felvehető szívek

Az utolsó említetlen pályaelem az életet adó szív (29. ábra: Szív). Ez általában nehezen megközelíthető helyen, bizonyos esetekben csak erősítés használatával kaparintható meg. Egy extra próbálkozás megmentheti a játékost a játék végétől, így keresésük erősen ajánlott.

29. ábra: Szív

### Csontvázak

#### Kardforgató

A kardforgató csontvázharcosok (30. ábra: Kardforgató csontvázharcos) gyorsan támadó és mozgó ellenfelek, akik pillanatok alatt a játékoshoz szökkennek, hogy aztán lesújtsanak rá pengéjükkel.

A képen koponya, rajzfilm, csont, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* Gyors közelharci támadások, melyek találatkor sebeznek és visszalöknek
* Sebzés: 20
* Életerő: 150
* Támadás sebessége: nagyon gyors
* Ha a játékost észreveszi, futni kezd felé, fokozatosan gyorsulva.

30. ábra: Kardforgató csontvázharcos

#### Pörölyös

A pörölyös csontvázharcosok (31. ábra: Pörölyös csontvázharcos) kinézetük ellenére igen fürgék, támadásaik pedig mindenáron elkerülendőek. Ha a játékos a pálya egy fontos mérföldkövéhez ér, szinte biztos lehet benne, hogy találkozni fog eggyel.

A képen képernyőkép, rajzfilm látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* Lassan végrehajtott közelharci támadások, melyek találatkor nagy sebzést és jelentős visszalökést okoznak
* Sebzés: 40
* Életerő: 250
* Támadás sebessége: igen lassú
* Ha a játékost észreveszi, futni kezd felé, fokozatosan gyorsulva

31. ábra: Pörölyös csontvázharcos

#### Távolsági

A távolsági csontvázharcosok (32. ábra: Távolsági csontvázharcos) lassabban mozognak, ám sűrűn indítanak távolsági támadásokat. Lövedékeik elkerülése különösen fontos, főleg a levegőben tartózkodó játékos számára. Ha például a játékos éppen a második ugrását használja, és eközben találatot kap, a visszalökés hatására könnyen elvétheti a célplatformot, és veszélyes felületre zuhanhat. A közelharc szintén körültekintést igényel: rosszul időzített támadás esetén az ellenfél elrúgja magától a játékost, ismét távolsági küzdelembe kényszerítve őt.

* A képen képernyőkép, koponya látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.Közepes időközönként folytatott távolsági támadások, melyek találatkor sebzést és visszalökést okoznak
* Lövedék sebzése: 10
* Lövedék sebessége: közepesen gyors
* Rúgás sebzése: 15
* Rúgás támadási ideje: nagyon gyors
* Életerő: 200
* Ha a játékost észreveszi, távolsági támadást indít ellene, majd gyorsabban mozog az irányába

32. ábra: Távolsági csontvázharcos

### Pályák

#### Tutorial

Új játék indításakor a játékos a bevezetőként szolgáló *Tutorial* pályán találja magát (33. ábra: Első betöltés). A pályán rögzített paneleken megjelenő instrukciók segítségével megismerkedhet a játék irányításával, a játékban szereplő különböző pályaelemekkel, erősítésekkel és ellenfelekkel.

A képen képernyőkép, szöveg, Digitális képszerkesztés, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

33. ábra: Első betöltés

#### Dungeon

A bevezető pályáról kiszabadulva, a játékos még mindig a kazamaták mélyén elhelyezkedő *Dungeon* szintre jut (34. ábra: Dungeon pálya). Ezen a pályán már nagyobb kihívást jelentő akadályokkal kell szembenéznie, és a bevezető pályához képest sokkal nagyobb terület felfedezésének néz elébe. A szint elrendezése vertikális hangsúlyú, tehát a kiindulópont és a kijárat között jelentős magasságbeli különbség van. A játékos itt már minden megtanult mechanikát alkalmazhat, és alkalmaznia is kell.

A képen szöveg, képernyőkép, Számítógépes játék, Digitális képszerkesztés látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

34. ábra: Dungeon pálya

#### Village

A kazamatákból kiszabadulva a játékos új környezetben találja magát (35. ábra: A Village szint bejárata). A föld alatt játszódó, rideg színezésű első szinthez képest a második szint színekkel teli és pozitívabb hangulatú.

35. ábra: A Village szint bejárata

A képen szöveg, képernyőkép, fa látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A *Village* szint (36. ábra: Village platformok) egy elhagyatott faluban játszódik, melyet csontvázharcosok szálltak meg. A korábbi pályához képest lineárisabb felépítésű, hosszan elnyúló terület várja a játékost, nehezebb akadályokkal berendezve. A ritkábban elhelyezett éledési pontok, a keskenyebb platformok és hosszabb csapdazónák fokozott precizitást követelnek, számos hibára adva lehetőséget. Jelenleg a játékban mindössze két szint érhető el, ám a rendszer könnyedén bővíthető további pályákkal.

A képen képernyőkép, Videojáték-szoftver, Számítógépes játék, 3D modellezés látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

36. ábra: Village platformok

### Győzelmi képernyő

Az utolsó pálya teljesítése után a felhasználó a győzelmi képernyőre kerül (37. ábra: Győzelmi képernyő). Itt megtekintheti a játék során összegyűjtött pontszámot, valamint a legyőzött csontvázak és széttört pályaelemek számát. A képernyő alján egyetlen gomb található *Back to Menu* felirattal, amely visszavezet a játék főmenüjébe. Innen a játékos csak új játékot indíthat, mivel a jelenlegi verzióban nincs lehetőség pályaválasztásra; újrajátszáskor a szinteket kizárólag a bemutató szintről, lineáris sorrendben lehet végigvinni.

A képen szöveg, képernyőkép, felhő látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

37. ábra: Győzelmi képernyő

# Fejlesztői dokumentáció

## Használt eszközök és technológiák

### Unity Engine választásának indoklása

A játék fejlesztéséhez a Unity játékmotort választottam, mert jelenleg az egyik legelterjedtebb és legsokoldalúbb eszköz ezen a területen. A Unity különösen jól használható 2D-s és 3D-s projektekhez is, de mivel én egy 2D platformjátékot készítettem, számomra főleg a kétdimenziós fejlesztéshez kínált eszközei voltak igazán hasznosak. Például a *Tilemap* rendszerrel sokkal gyorsabban tudtam megépíteni a pályákat, az *Animator* segítségével pedig a karakterek és a különböző játékelemek animációját kezeltem egyszerűen.

A Unity másik nagy előnye, hogy multiplatform támogatással rendelkezik. Ez annyit jelent, hogy a játékot szinte különösebb változtatás nélkül ki lehet adni Windowsra, Mac-re, mobilokra vagy akár konzolokra is. Ráadásul már a fejlesztés elején is lehetőség van gyors prototípusokat készíteni, amit később könnyen lehet optimalizálni és véglegesíteni, hála a motor erős erőforrás-kezelési megoldásainak.

**Fő előnyök, amiket tapasztaltam:**

* **Multiplatform támogatás:** ugyanaz a projekt könnyedén futhat többféle eszközön.
* **Gyors fejlesztés:** a sok beépített eszköznek köszönhetően gyorsan lehet haladni a munkával, legyen szó pályatervezésről vagy játékmenet fejlesztésről.
* **Kifejezetten jó 2D támogatás:** a Unity erre is komoly hangsúlyt fektetett, ami egy platformjáték esetében óriási előny volt.

**Hátrányok:**

* Bár a Unity nagyon jó eszköz 2D játékok fejlesztésére, elsősorban 3D-re optimalizált motor, így a 2D-s játékok is „3D környezetben” futnak.
* Ez néha indokolatlanul magas erőforrás-használathoz vezethet (CPU, RAM, GPU), különösen mobilokon vagy gyengébb gépeken.

Mielőtt a Unity mellett döntöttem, megnéztem más lehetőségeket is**:**

* **Godot Engine:** Ez egy könnyen kezelhető, ingyenes motor, ami szintén jól működik 2D projektekhez. Viszont a kisebb közösség miatt nehezebb megoldásokat találni, ha valami problémába ütközünk.
* **Unreal Engine:** Technológiailag nagyon erős, főleg 3D-ben. De épp emiatt túlzás lett volna egy egyszerűbb 2D-s platformerhez, ráadásul a kezelése is jóval bonyolultabbnak tűnt számomra.

Összességében azért maradtam a Unitynél, mert egyszerre adott gyors fejlesztési lehetőséget, erős 2D támogatást és a multiplatform megjelenést is megkönnyítette. Ezek együtt olyan előnyök voltak, amik a projektemhez tökéletesen passzoltak.

### C# programozási nyelv választásának indoklása

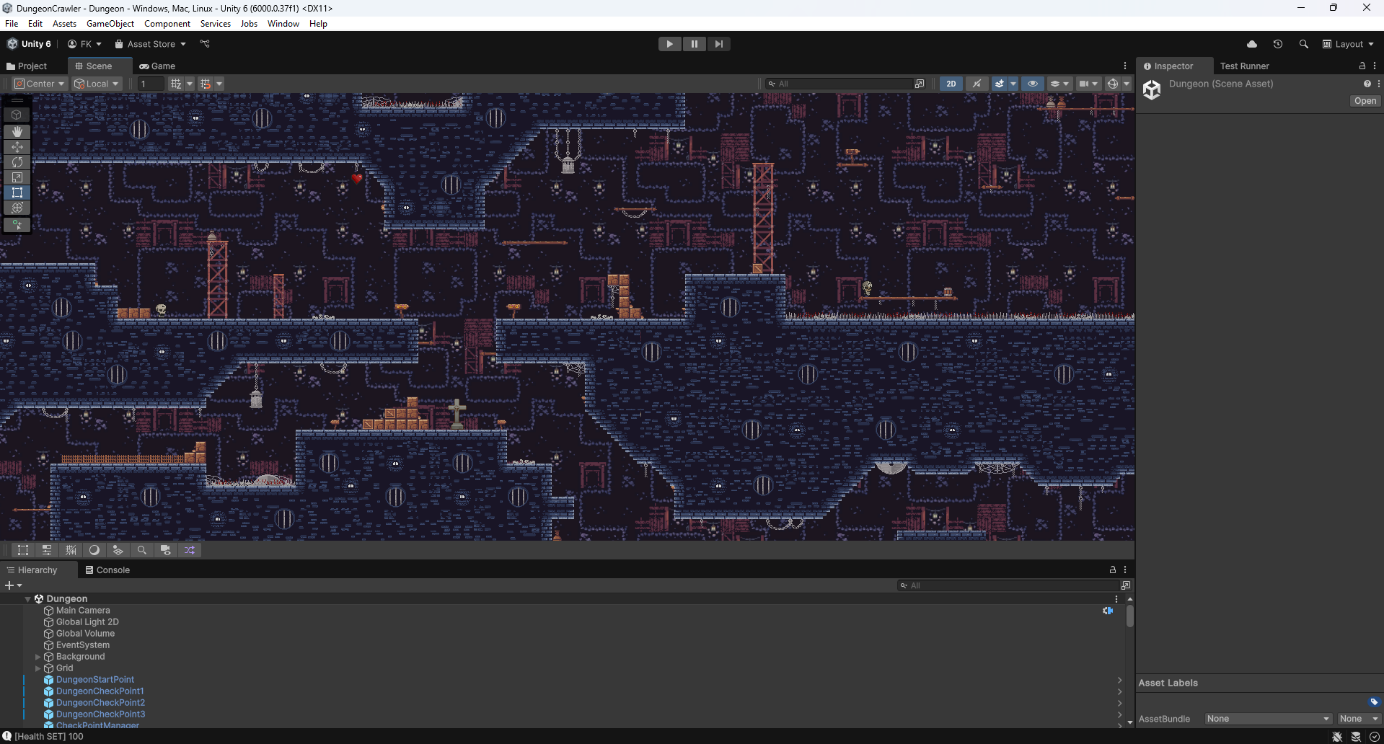
A fejlesztés során C#-ot használtam, mivel a Unity alapból erre a nyelvre épül. A C# egy erőteljes, objektumorientált nyelv, ami jól illik a játékfejlesztéshez. A szintaxisa egyszerű, könnyen átlátható, ráadásul rengeteg hasznos könyvtárral és előre megírt megoldásokkal szolgál. A legnagyobb előnye az volt, hogy nagyon jól működik együtt a Unity motorral, így minden funkciót könnyedén elértem vele.

### A dolgozathoz használt fontos Unity fogalmak

A Unity-ben a fejlesztési folyamat egyik alapköve a **SceneView** és a **GameView** használata.

#### SceneView

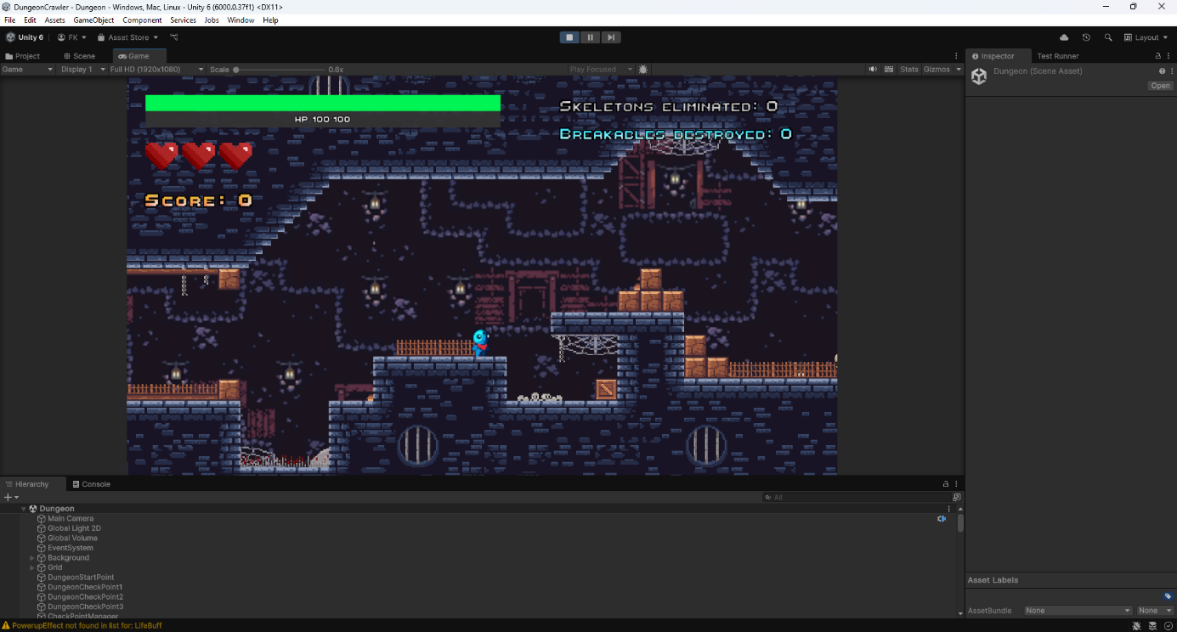
A *SceneView* (38. ábra: SceneView) lényegében a "szerkesztői nézet" – itt tudjuk kézzel összeállítani a jeleneteket, elhelyezni az objektumokat, finomhangolni a világot. Olyan, mint egy műhelyasztal, ahol még szabadon pakolgathatunk bármit, mielőtt ténylegesen "élesbe" menne.



38. ábra: SceneView

#### GameView

A 39. ábrán látható *GameView* („*Play* mód”) pedig az, ahol futtatni tudjuk a játékunkat közvetlenül a szerkesztőből. Itt látjuk, hogyan viselkednek az objektumok, hogyan működnek a scriptjeink éles környezetben, tehát valós idejű tesztelési lehetőséget ad. Fontos, hogy amit *Play* módban változtatunk (pl. mozgatunk egy objektumot), az nem marad meg, ha kilépünk a *Play* módból - ilyenkor a szerkesztő visszaáll az eredeti állapotba.



39. ábra: GameView

#### MonoBehaviour

A Unity-ben a legtöbb script alapja a *MonoBehaviour* osztály. Ez az, amit akkor használunk, amikor egy scriptet közvetlenül egy *GameObject*hez akarunk kötni. Például, ha egy karakter mozgását, egy platform viselkedését vagy egy ellenség mesterséges intelligenciáját akarjuk vezérelni, akkor *MonoBehaviour*-t öröklünk.

#### ScriptableObject

A *ScriptableObject* ezzel szemben egy kicsit más jellegű. *ScriptableObjectek*et akkor érdemes használni, amikor adatok tárolására van szükségünk függetlenül attól, hogy azok éppen melyik *GameObject*hez tartoznak. Például karakterstatisztikákat, fegyverek adatait vagy pálya konfigurációkat lehet így jól strukturálni. Nagy előnye, hogy memóriában hatékonyabb, és egyszerre több *GameObject* is hivatkozhat ugyanarra a *ScriptableObject* példányra.

Röviden:

* *MonoBehaviour* = "viselkedést" ad egy *GameObject*hez
* *ScriptableObject* = "adatot" ad függetlenül a jelenet objektumaitól

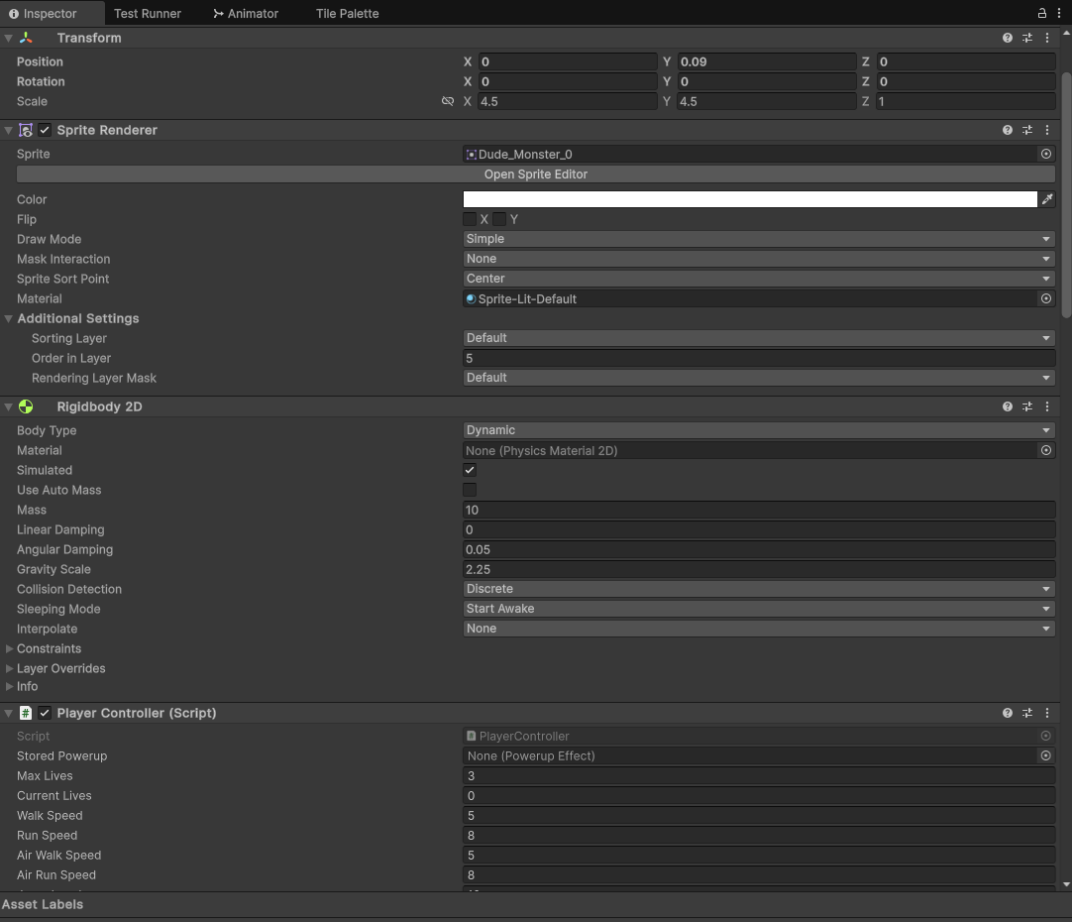
#### Alapvető metódusok

A Unity scriptjeiben többféle "életciklus" metódus is rendelkezésre áll, amikkel pontosan meg lehet határozni, mikor mit akarunk csinálni.

* **Awake()**: Akkor fut le, amikor az objektum a játék indításakor betöltődik. Itt kell beállítani a legfontosabb kezdeti értékeket.
* **Start()**: Ez már csak akkor hívódik meg, amikor az objektum aktívvá válik a játékban. Bizonyos kezdeti értékeket érdemes lehet csak itt beállítani az *Awake* helyett.
* **Update()**: Ez a játék minden egyes képkockájánál lefut. Ide kell tenni azokat a kódrészeket, amik folyamatosan figyelnek valamit, például a játékos inputját.
* **FixedUpdate()**: Ez hasonló az *Update*-hez, de nem a képkockák számához, hanem fix időlépésekhez igazodik. Főleg fizikával kapcsolatos számításokhoz ajánlott használni, mert itt megbízhatóbb az időzítés.

#### Inspector komponens

Az **Inspector** (40. ábra: Inspector menü) egy kulcsfontosságú szerkesztőpanel a Unity Editoron belül, amely lehetővé teszi az egyes *GameObjectek*hez rendelt komponensek és azok tulajdonságainak megtekintését és szerkesztését. Amikor kiválasztunk egy objektumot a Hierarchy nézetben, az Inspector panel automatikusan megjeleníti az ahhoz az objektumhoz kapcsolt összes komponenst - például a *Transform*ot, *Rigidbody*-t, *Collider*-t vagy bármely egyedi *MonoBehaviour* scriptet. *Az Inspector* segítségével tehát vizuálisan, kódírás nélkül módosíthatjuk egy objektum viselkedését és megjelenését.

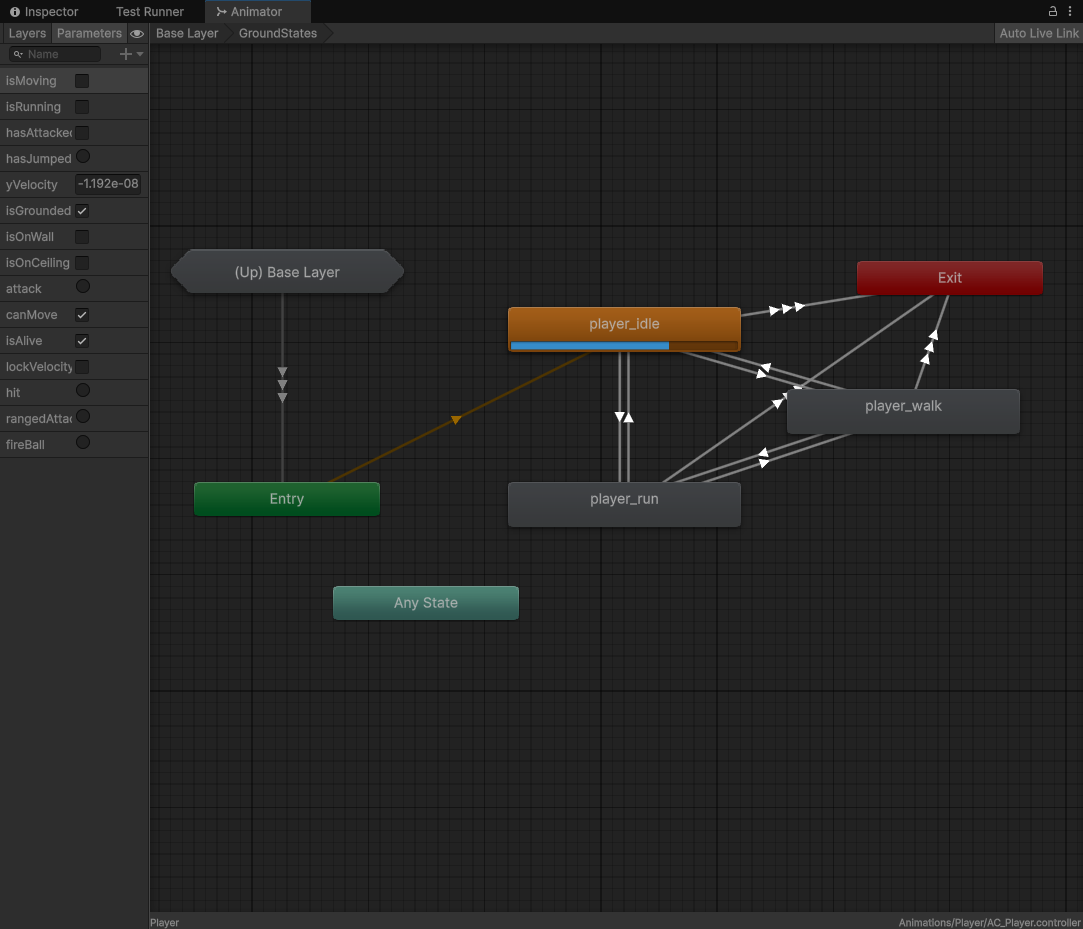


40. ábra: Inspector menü

#### Animator komponens

Az **Animator** (41. ábra: Animator komponens) a Unity animációs rendszerének központi része. Ezzel tudjuk irányítani a karakterek és más játékelemek animációit, például egy futás, ugrás, vagy támadás mozgásait.

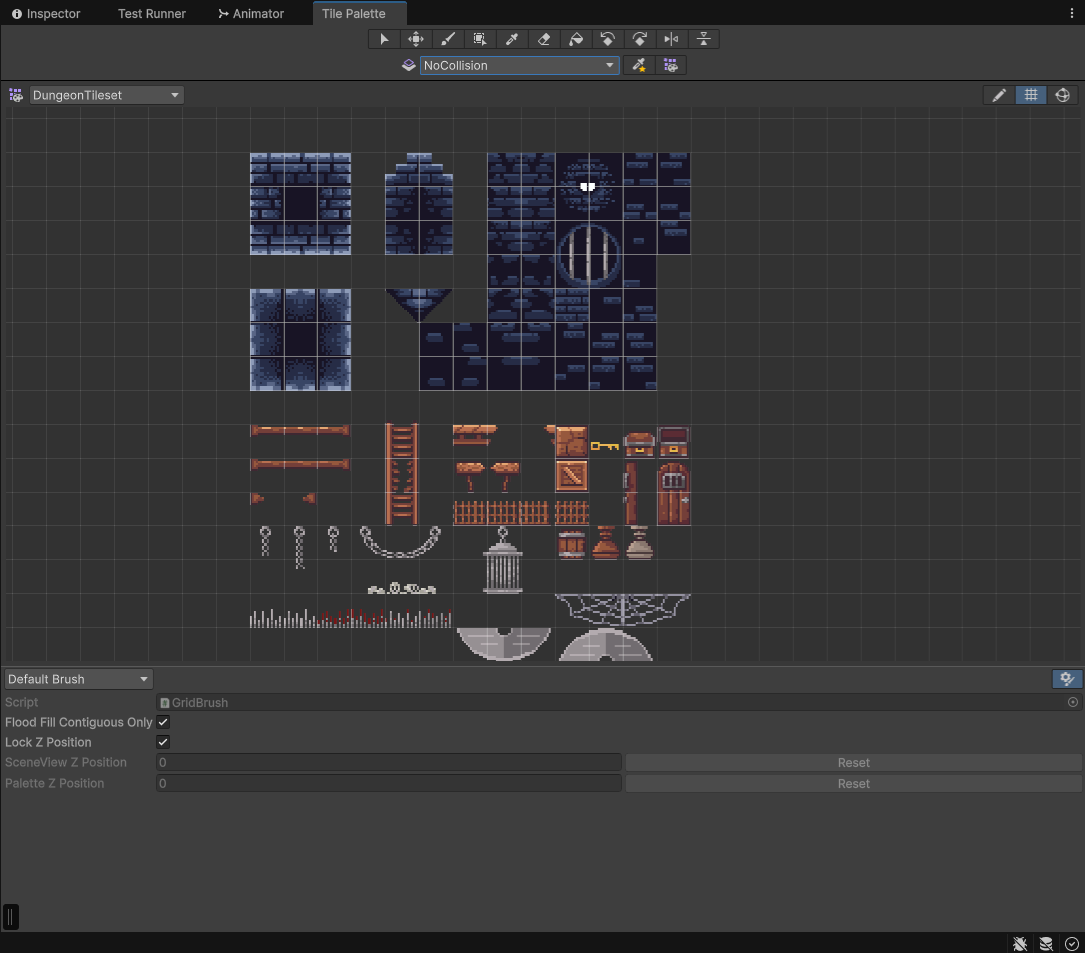
Az *Animator*ban állapotgépek használatával meghatározzuk, hogy az egyes animációk között milyen feltételekkel lehet váltani. Például, ha a játékos "futás" állapotban van, és megnyom egy gombot, akkor az *Animator* átválthat "ugrás" állapotba. A rendszer vizuálisan is jól áttekinthető: húzogatni lehet a kapcsolatokat az animációk között, és be lehet állítani különféle paramétereket (pl. sebesség, gombnyomás), amik alapján dönt, hogy mikor váltson.



41. ábra: Animator komponens

#### Tilemap és Tile Palette komponens

A **Tilemap** egy elképesztően hasznos eszköz, főleg 2D-s játékoknál. Gyakorlatilag egy rácshálóra tudunk kis csempéket (*tile*-öket) "rajzolni", amiből gyorsan és egyszerűen lehet pályákat összerakni. A *Tilemap* rendszer nagyon erős abban, hogy nagyobb, részletesebb pályákat is könnyen kezelhetővé tesz. Nem kell minden kis faldarabot, padlót vagy akadályt külön objektumként elhelyezni, elég egyszerűen a **Tile Palette**-ből (42. ábra: Tile Palette komponens) kiválasztani az adott csempét, és "kifesteni" vele a pályát. Ráadásul a *Tilemap*-hez könnyen hozzárendelhetünk *collider* rétegeket is, így a játékos vagy az ellenségek rögtön érezni fogják a világ határait anélkül, hogy kézzel kellene mindenhol egyesével *collider* komponenseket feltenni. Ennek hiánya a fejlesztés nehézségét hatványozta volna, így örömmel használtam ki a komponens nyújtotta lehetőségeket.



42. ábra: Tile Palette komponens

#### A Unity beépített fizikai rendszere

A Unity beépített fizikai rendszere (*Physics System*) gondoskodik arról, hogy a játékban szereplő objektumok "valóságosan" mozogjanak, ütközzenek, reagáljanak az erőhatásokra.

Két nagy része van: **2D Physics** (pl. *Rigidbody2D*, *Collider2D*) és **3D Physics** (pl. *Rigidbody*, *Collider*). A projekt során a *2D* *Physics* rendszer használatára volt szükségem.

Amikor egy objektumhoz *Rigidbody2D* komponenst adunk, akkor az bekerül a fizikai motor számításai közé, tehát például súlya lesz, leesik a ráható gravitációs erő miatt, vagy nekiütközik más dolgoknak (*collide*-ol). A physics számítások a *FixedUpdate* során frissülnek, hogy a fizikai interakciók stabilak és kiszámíthatóak legyenek. A *collider* komponensek pedig meghatározzák, hogy milyen formája és mérete van az objektumnak fizikai szempontból. Az egész rendszer célja az, hogy a fejlesztőnek ne kelljen kézzel kiszámolnia ütközéseket, gyorsulásokat, forgásokat - ezt a motor automatikusan megoldja, rengeteg időt megspórolva. Úgy gondolom, hogy az effektív programozás egyik legfontosabb eleme az a belátás, hogy vannak dolgok, amiket nem érdemes „újra feltalálni”. Véleményem szerint az alapvető fizikai hatások rendszere jelen projekt esetében tökéletes példának számít.

## Felhasználói esetek

## A játék fontosabb scriptjei (megvalósítási terv)

* egy részét holnap?

## A scriptek kapcsolatai (megvalósítási terv)

* egy részét holnap?

## A játék tesztelése

### Manuális tesztelés

A *Dungeon Crawler* készítése alatt több ismerősöm is kipróbálta a játékot, különböző fejlesztési fázisokban. Ezen tesztelések számos hibára rámutattak, lehetővé téve a gyors javításokat. A legelső komolyabb hiba a projekt elején került felszínre, a játékos alapvető irányításainak implementálásakor. Amikor a játékos oldalról egy vékony platform széléhez ugrott, - amely vastagsága kisebb volt magánál a játékosnál is - vertikális sebessége azonnal felülíródott és se nem zuhant, se nem emelkedett tovább, mintegy folyamatosan „zuhanva”. Ennek oka a polírozatlan *hitbox*, és *collider*-kezelésem volt, pontosabban az, hogy az ütközést helytelenül, csupán a játékos feje és lába körül vizsgáltam *Raycast*ek használatával. Ebből kifolyólag, ha egy platform ezen vonalak közé esett, a játékos nem detektálta az ütközést vele, ez pedig kódbéli problémákat okozott a játékfizika kezelésénél. A megoldást egyszerűen sűrűbb *Raycast* használat jelentette. Még egy hiba, melyet egy barátom talált meg manuális tesztelés során: amikor a játékos egy ellenfél fejére elég precízen érkezett ugrást követően, akkor tovább utazhatott a csontváz fején, miközben az tanácstalanul próbálta őt megtámadni. Erre egy igen egyszerű megoldást találtam: az ellenfelek fejére egy *collider*-t helyeztem, mely a játékos *collider*-ével érintkezve - jelképes sebzés kíséretében - visszataszította a játékost, eltántorítva őt a hasonló próbálkozásoktól. Ez később más szempontból is egy jó változtatásnak bizonyult, ugyanis így a pálya alacsonyabb magasságú területein elhelyezett csontvázak megkerülése, átugrása immár nagyobb kihívássá vált a felhasználónak.

### Automatizált tesztelés Unit Testekkel

A Unity-ben végzett unit tesztelés során két jól elkülönülő típusra támaszkodhattam: **EditMode** és **PlayMode** tesztekre. Mindkettő más célra való, és a használatuk során én is gyorsan megtapasztaltam, hogy mikor melyik a hasznosabb.

Az *EditMode* tesztek inkább a nem-jelenetfüggő működés ellenőrzésére alkalmasak. Ilyenkor nincs szükség a játék tényleges futtatására vagy idő-alapú események szimulálására - egyszerűen példányosítunk egy *GameObject*et, hozzárakunk néhány komponenst, beállítjuk a kezdeti értékeket, és ellenőrizzük a viselkedésüket. Például a *Skeleton* osztályom esetében így tudtam gyorsan letesztelni, hogy egy irányváltás valóban megfordítja-e a karaktert, vagy hogy a *knockback* logika helyesen változtatja-e meg a *Rigidbody2D* sebességét.

A *PlayMode* tesztek viszont már magát a játékot, annak időbeli viselkedését is figyelembe veszik - olyasmiket lehet velük vizsgálni, amik csak akkor történnek meg, ha a játék valóban fut. Itt teszteltem többek között a *Damageable* komponensemet is: hogyan reagál egy találatra, miként aktiválódik az *invincibility* mechanika, vagy hogy a karakter halála után ténylegesen leáll-e a sebződés. Itt fontos, hogy az *IEnumerator* alapú tesztelés lehetővé teszi a Unity *frame*-alapú működésének szimulálását, amit az *EditMode* nem tud.

A tesztek egyszerű futtatására a *Test Runner* komponens ad lehetőséget (43. ábra: Test Runner komponens).

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

43. ábra: Test Runner komponens

# Összefoglalás

Szakdolgozatom írásába bármiféle előzetes Unity tapasztalat nélkül vágtam bele. A motor alapvető komponensei, mint például a jelenetek (*Scenes*) vagy az ezekhez tartozó *GameObject*ek működése, kezdetben teljesen ismeretlen volt számomra. Az első pár indításnál a kezelőfelület használata is nehézséget okozott, mivel a Unity felépítését azelőtt kizárólag csak képernyőképeken keresztül ismertem. Ennek következtében eleinte bizonytalanság és tanácstalanság jellemezte a munkámat. Ez az érzés azonban viszonylag gyorsan elmúlt. Ahogy egyre több időt töltöttem a motor megismerésével, fokozatosan világossá vált számomra, hogy az alapvető működések elsajátítása nem igényel lehetetlen erőfeszítést. Rövid időn belül magabiztosabban tudtam navigálni a különböző nézetek és eszközök között, ami lehetővé tette számomra a gyorsabb haladást.

A *Dungeon Crawler* fejlesztése során kiemelt szerepet kaptak az objektum-orientált programozási (OOP) elvek. Bár a projekt nem minden osztályánál sikerült teljes mértékben megvalósítanom az ideális gyakorlatokat, a fejlesztési folyamat előrehaladtával egyre tudatosabban törekedtem ezek alkalmazására. Így például az enkapszuláció, az egységesség és az újrafelhasználhatóság fontossága világosan kirajzolódott számomra. Az implementált checkpoint rendszer, a mentési mechanizmus, valamint a karakterek és interaktív pályaelemek kezelése során különösen érzékelhető volt az OOP megközelítés előnye.

Összegzésként elmondható, hogy bár a végső kódbázis több területen hagy maga után fejlesztési lehetőségeket, a projekt során megszerzett tapasztalatok jelentősen hozzájárultak az objektum-orientált szemléletem és gyakorlati tudásom fejlődéséhez. A szakdolgozatnak köszönhetően nemcsak technikai ismereteim bővültek jelentősen, hanem megerősödött az elhatározásom is, hogy a jövőben további játékfejlesztési projektekkel foglalkozzak Unity-ben. Célkitűzésem, hogy a jövőbeli munkáim során következetesebben alkalmazzam az objektum-orientált programozási, clean code, valamint szoftvertervezési elveket, ezzel is tovább növelve a fejlesztések minőségét és hatékonyságát.

# Fejlesztési lehetőségek

A játék jelenlegi verziója már biztos alapot ad a jövőbeli fejlesztésekhez, ugyanakkor számos terület van, ahol a kód tisztábbá, modulárisabbá tételével és a játékélmény gazdagításával még többet kihozhatunk belőle. Az alábbiakban néhány javaslatot fogalmazok meg, amelyek az OOP, clean code elvek és a játékfejlesztési legjobb gyakorlatok figyelembevételével segíthetik a jövőbeli bővítéseket.

**1. Kódrefaktorálás és OOP alapú fejlesztések**

A játék kódja számos területen javítható az objektumorientált programozás (OOP) alapelveinek alkalmazásával. Egy ilyen irányú fejlesztés segíthet abban, hogy a jövőbeni bővítések és karbantartások könnyebbek legyenek. Az **egységes interfészek** használata, mint például az IDataPersistence, már jelen van, de az interfész további bővítése és az osztályok megfelelő szétválasztása még jobban elősegítheti az új funkciók gyors implementálását. A **Factory Design Pattern** alkalmazása például a power-up-ok kezelésére, különösen a Breakable osztályokban, lehetőséget adna új erősítések könnyű hozzáadására anélkül, hogy a meglévő kódot módosítani kellene. Továbbá, az **Event-Driven Programming** alkalmazása, amely az események jobb kezelése révén növelné az osztályok modularitását, szintén elősegíthetné a jövőbeli mechanikák hozzáadását.

**2. Clean Code és kódminőség javítása**

A kód minősége szoros összefüggésben áll a jövőbeli karbantartással, és a clean code elvek alkalmazása segíthet a kód hosszú távú fenntarthatóságában. A **kód duplikáció csökkentése** érdekében érdemes lenne egy központi tárgykezelő rendszert kialakítani, amely minden típusú tárgy kezelésére alkalmas. Ezzel a különböző pickup osztályok közötti kódismétlődéseket minimalizálhatjuk. Emellett a **tiszta metódusok** alkalmazása is fontos: az olyan hosszú és bonyolult metódusok, mint az OnTriggerEnter2D, könnyen szétválaszthatók kisebb, jól definiált segédfunkciókra, így a kód karbantartása és érthetősége jelentősen javulhatna. A **változónevek** egyértelműsítése szintén fontos, például a hasBeenPickedUp helyett érdemes olyan nevet választani, ami jobban tükrözi a változó célját, mint például isPowerupCollected.

**3. Újrafelhasználhatóság és bővíthetőség**

A jövőbeli fejlesztésekhez fontos, hogy a kód bővíthető és újrafelhasználható legyen. A **moduláris felépítés** érdekében célszerű lenne a különféle pickupok és breakable objektumok kezelését jobban elválasztani, így új funkciók könnyedén hozzáadhatók lennének. A **power-up rendszer fejlesztése** is elengedhetetlen: egy központi osztály létrehozása, amely kezeli az összes erősítést, segítene a különböző típusú power-up-ok egyszerűbb hozzáadásában. Így nem kellene minden új típusú power-up-ot a különböző osztályokban manuálisan implementálni.

**4. Játékfejlesztési bővítések**

A játék alapvető mechanikái már most is erősek, de a jövőben számos új funkcióval gazdagítható. Az **animációk** és **zene** bevezetése jelentősen növelné a játék élvezetét. A karakterek és ellenségek animálása, valamint a pályákhoz illő zenei aláfestés mind hozzájárulnának a játék atmoszférájának fokozásához. A jövőben további **pályák** és **környezeti elemek** hozzáadása még változatosabbá tenné a játékot. Az új pályák hosszának és összetettségének növelése mellett új kihívások, például rejtvények is bevezethetők lennének. Továbbá, egy **többjátékos mód** hozzáadása, amely lehetővé tenné a kooperatív játékot, új dinamikát és közösségi élményt adhatna a játéknak.

Összességében a játék alapja már most is stabil, de számos lehetőség rejlik a kód és a játékmenet továbbfejlesztésében, amelyek még gazdagabbá és élvezetesebbé tehetik a játékot a jövőben. A javasolt fejlesztések lehetővé teszik, hogy a játék még könnyebben bővíthető és karbantartható legyen, miközben új funkciók is könnyedén integrálhatók.

# Melléklet és felhasznált források

A játék fejlesztése során számos Unity-csomagot, valamint külső, ingyenesen felhasználható grafikai és audióelemeket használtam fel, amelyeket ebben a pontban dokumentálok.

## Felhasznált Unity csomagok

Itt csak a legfontosabb csomagokat említem meg, amelyeket a projekt során használtam, és amelyek nem feltétlenül kerülnek automatikusan telepítésre egy hasonló projekt létrehozásakor.

* **2D Animation**: A 2D-s animációk kezelésére szolgáló csomag. Segítségével minden játékon belüli interakció sokkal életszerűbbé tehető. Például, ha egy karakter ugrik vagy támad, a csomag lehetővé teszi az animációk dinamikus, zökkenőmentes lejátszását.
* **2D Sprite**: A 2D grafikák, úgynevezett *sprite*-ok importálására és manipulálására használjuk. E nélkül nem lenne lehetséges a játék vizuális megjelenésének kialakítása.
* **2D Tilemap Editor**: Segít gyorsan és egyszerűen létrehozni a rács alapú pályákat és térképeket. A *tilemap* alapú pályaépítéshez elengedhetetlen csomag.
* **Input System**: Lehetővé teszi az összes felhasználói bemenet, például a billentyűzet, egér vagy kontroller személyre szabható kezelését.
* **Cinemachine**: A kamerakezelést könnyíti meg, automatikusan és dinamikusan irányítja a jelenetekben a kamerát. Maximálisan személyre szabható kameraviselkedést biztosít.
* **Custom NUnit**: Tesztelési eszközként szolgál, biztosítva, hogy a projekt kódja hibamentesen működjön, és segít a tesztelési folyamatok integrálásában. Ha például egy új funkciót adunk hozzá, mint mondjuk egy új karakter képességet, a Custom NUnit segít ellenőrizni, hogy az új kód nem okoz-e problémát más rendszerekben.
* **Test Framework**: Még egy tesztelési eszköz, mely lehetővé teszi, hogy a különböző teszteket egyszerűen végrehajtsuk. Megfelelően strukturált projekt és a tesztfájlok helyes elhelyezése esetén egyetlen gombnyomással elvégezhetjük vele a tesztelést.

## Grafikus anyagok

A játékban megjelenő betűtípus (*Thaleah*) forrása:

<https://assetstore.unity.com/packages/2d/fonts/free-pixel-font-thaleah-140059?srsltid=AfmBOopMM5TR1w9viZNem63g-vcO3zVXKf64_L4wJfBQl9JnWPYcGZ1k>

A játékban felhasznált további pixeles grafikák forrásai:

* Az irányított karakter, lövedék asseteinek forrása: <https://free-game-assets.itch.io/free-tiny-hero-sprites-pixel-art>
* A csontvázak asseteinek forrása: <https://craftpix.net/freebies/chibi-skeleton-warrior-character-sprites/?num=3&count=81&sq=skeleton&pos=12>
* Kulcshoz felhasznált assetek: <https://karsiori.itch.io/pixel-art-key-pack-animated>
* Tűzlabdához felhasznált assetek: [https://bdragon1727.itch.io/fire-pixel-bullet-16x16](https://bdragon1727.itch.io/fire-pixel-bullet-16x16%20)
* Erősítések pixel art-ja: <https://assetstore.unity.com/packages/2d/gui/icons/2d-potions-pixel-art-196023>
* Szív UI és pályaelem forrása: <https://fliflifly.itch.io/hearts-and-health-bar>
* Tutorial és Dungeon pálya építéséhez felhasznált assetek, kijáratként szolgáló ajtó és a széttörhető elemek forrása: <https://incolgames.itch.io/dungeon-platformer-tile-set-pixel-art>
* Tutorial és Dungeon pálya háttere, Dungeon pályán megtalálható emelvények forrása: [https://grafxkid.itch.io/cave-tileset](https://grafxkid.itch.io/cave-tileset%20)
* Tutorial pályán megtalálható instrukciós panelekhez felhasznált assetek: <https://bdragon1727.itch.io/custom-border-and-panels-menu-all-part>
* Village pálya építéséhez felhasznált assetek: <https://cainos.itch.io/pixel-art-platformer-village-props>
* Village pálya és Győzelmi képernyő háttere: <https://free-game-assets.itch.io/free-horizontal-game-backgrounds>

## Hang és zenei anyagok

A játékban hallható hangeffektek (*SFX*) egyik forrása: <https://elevenlabs.io/>.

* Az ElevenLabs egy mesterséges intelligenciára épülő vállalat, amely élethű, érzelmekkel teli mesterséges hangokat készít szövegfelolvasás, hangklónozás és automatikus szinkronizálás segítségével
* Felhasználás: széttörhető elemek széttörésénél lejátszott hangeffekt, tűzlabda, kő lövedék hangeffektjei, csontvázharcosok sebződésének és halálának hangeffektje, játékos sebződésének és halálának hangeffektje

Egy másik forrás, ahonnan ingyenesen felhasználható hangeffekteket töltöttem le:

* <https://pixabay.com/sound-effects/>
* Felhasználás: a fent említett hangeffektek kivételével minden innen ered

A játék háttérzenéinek forrásai:

* Main Menu/Tutorial jelenet: <https://www.fesliyanstudios.com/royalty-free-music/download/8-bit-menu/287> (szerző: David Renda)
* Dungeon szint: <https://www.fesliyanstudios.com/royalty-free-music/download/retro-platforming/454> - Slower Tempo - (szerző: David Fesliyan)
* Village szint: <https://www.fesliyanstudios.com/royalty-free-music/download/8-bit-retro-funk/883> (szerző: David Renda)

## Egyéb források

A fejlesztés során a Unity hivatalos dokumentációján kívül, más segédanyagot is használatba vettem, méghozzá a következő Youtube-csatornákat, melyek segítségével gyorsan megtanulhattam új Unity-s technikákat és megoldásokat.

* Infallible Code (<https://www.youtube.com/@InfallibleCode>)
* Rehope Games (<https://www.youtube.com/@RehopeGames>)
* Shaped by Rain Studios (<https://www.youtube.com/@ShapedByRainStudios>)
* Pandemonium (<https://www.youtube.com/@PandemoniumGameDev>)