Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Hallgató:
Török Dániel

2025

Szegedi Tudományegyetem
Informatikai Tanszékcsoport

2D souls-like platformer játék

Szakdolgozat

Készítette:
Török Dániel
programtervező
informatikus szakos
hallgató

Témavezető:
Jász Judit Dr.
adjunktus

Szeged
2025

FELADATKIÍRÁS

A szakdolgozat célja egy 2D-s platformer souls-like videójáték fejlesztése, amelyben a játékos egy nyitott világban tud felfedezni, illetve ellenségekkel küzdeni, miközben fejleszti a karakterét, tárgyakat gyűjt és készít, képességeket old fel, amikkel később további részeket old fel a nyitott világból. Fontos, hogy a játék állást el lehessen menteni, továbbá betölteni. A fejlesztési rendszernek hasonlítania kell egy souls-like játékban lévő fejlesztési rendszerhez, vagyis legyenek a játékosnak leíró statisztikái (például: strength – erő stb.…) amik szintenként eggyel növelhetőek, és a szintlépéshez szükséges erőforrás szintenként növekedjen. Szükséges még platformer játékokra hajazó elemek beépítése is, például: fal ugrás, dupla ugrás, platformok, egyirányú platformok stb.…, illetve mivel nyitott a világ, és akár elég nagy is lehet, ezért érdemes egy gyors utazási mechanikát beépíteni, hogy a térkép gyorsan navigálható legyen. Ellenségek, illetve egy főellenség elkészítése is prioritás, ez egy fő szempontja a souls-like játékoknak, az ellenségek ki egyensúlyozásával egyenértékűen, ugyanígy nagyon fontos az ellenfelek – kivéve a főellenség – visszaállítása ellenőrző pont interakció által. Továbbá csapdák, használható objektumok implementálása is része a feladatnak.

TARTALMI ÖSSZEFOGLALÓ

- ***A téma megnevezése***

2D souls-like platformer játék a Godot játék motorban

- ***A megadott feladat megfogalmazása***:

Egy olyan videójáték fejlesztése, amely lehetővé teszi a játékos számára, hogy fejlessze a karakterét, tárgyakat készítsen, ellenségekkel küzdjön, továbbá a játék tartalmazzon platformer elemeket (pl.: fal ugrás, dupla ugrás, platformok), a karakter halálakor a világban levő ellenfelek álljanak vissza (kivétel a főellenség(-ek)), egy rendszer, amely lehetővé teszi a játékos számára, hogy teleportálhasson előre meghatározott helyekre. Fontos funkciók fejlesztése például a játék állás mentése, játék betöltése.

- ***A megoldási mód:***

Az játék fejlesztése Godot játék motor segítségével történt. A pixeles képek megrajzolásához a Pixeloroma program lett használva, audió fájlok a freesound.org webhelyről lettek beszerezve, verzió kezeléshez a github.com lett használva. A pálya, illetve minden más egyéb fájl a játék motoron belül lett elkészítve.

- ***Alkalmazott eszközök, módszerek:***

Pixelorama

Godot engine 4.2, 4.3, 4.4, 4.5

GDScript beépített programozási nyelv

- ***Elért eredmények:***

Az elkészült játék lehetővé teszi a játékos egy előre elkészített pálya felfedezését, amely több részre van bontva, közben ellenségekkel küzdjön meg beleértve a főllenség(ek)-et, csapdákat kerüljön ki, fejlessze karakterét, tárgyakat hozzon létre/ gyűjtsön össze, elmentse a játék állását, betöltse a játék állását, teleportálhasson előre meghatározott pozíciókra. Az ellenfelek visszaállításra kerülnek a játékos halálakor. Platformer elemek implementálásra kerültek.

- ***Kulcsszavak:***

Godot, GDScript, souls-like, souls, platformer, tárgyak, játék, videojáték, nyitott világ

TARTALOMJEGYZÉK

BEVEZETÉS

A videojáték-fejlesztés az elmúlt évtizedekben az informatikai ipar egyik legdinamikusabban fejlődő területévé vált. A játékkészítés mára nem csupán szórakoztatóipari tevékenység, hanem komplex szoftverfejlesztési folyamat, amely ötvözi a programozást, a grafikai tervezést, a hangdizájnt és a mesterséges intelligenciát. A játékfejlesztő eszközök és motorok fejlődése lehetővé tette, hogy kis fejlesztőcsapatok vagy akár egyéni alkotók is professzionális minőségű játékokat hozzanak létre.

A szakdolgozat célja egy 2D, úgynevezett „souls-like" platformer játék megvalósítása a Godot Engine segítségével. A „souls-like" kifejezés olyan játékstílust jelöl, amelyet a Dark Souls sorozat inspirált: a jellemzői közé tartozik a magas nehézségi szint, a pontos időzítést igénylő harcrendszer, valamint a fokozatosan felfedezhető, összefüggő játéktér. A cél egy olyan prototípus létrehozása, amely hűen visszaadja ennek a műfajnak a hangulatát és mechanikai elemeit, ugyanakkor bemutatja a Godot Engine fejlesztési lehetőségeit és hatékonyságát [1, 2, 7, 16].

A választás indoka kettős: egyrészt a Godot Engine nyílt forráskódú és szabadon felhasználható, így ideális környezet az oktatási és kísérleti célú fejlesztésekhez; másrészt a souls-like műfaj összetett játékmenete kiváló lehetőséget biztosít a különböző játékrendszerek — például harc, mesterséges intelligencia, fizika és felhasználói interfész — integrálásának bemutatására. A projekt során a hangsúly a játékmenet logikai felépítésén, a karaktervezérlésen, az ellenségek viselkedésén és a pályarendszer kialakításán lesz.

A dolgozat a fejlesztési folyamat elméleti és gyakorlati aspektusait egyaránt tárgyalja. Az első fejezet a Godot játékmotor működését, felépítését és programozási környezetét mutatja be. Ezt követően részletesen ismertetésre kerül a játék tervezése, az implementáció folyamata, valamint az alkalmazott technológiák. A záró fejezet a fejlesztés során szerzett tapasztalatokat, valamint a lehetséges továbbfejlesztési irányokat foglalja össze.

1. TECHNOLÓGIÁK BEMUTATÁSA

1.1. Programnyelvek, technológiák

1.1.1. A Godot engine bemutatása

A **Godot Engine** egy modern, nyílt forráskódú játékmotor, amelyet Juan Linietsky és Ariel Manzur fejlesztett ki, és 2014-ben jelent meg az első stabil változata. A motor célja egy egységes, platformfüggetlen fejlesztői környezet biztosítása 2D és 3D játékok készítéséhez. A Godot alapja **C++ nyelven** íródott, és **GDScript** nevű, kifejezetten a játékfejlesztéshez tervezett, magas szintű szkriptnyelvet használ. A projektet a **Godot Foundation** és egy aktív közösség tartja fenn, amely folyamatos frissítésekkel és kiegészítésekkel biztosítja a rendszer fejlődését és hosszú távú támogatását [1, 6].

A Godot a **jelenet- és csomópontalapú architektúrára** épül, amely moduláris felépítést és magas fokú újra felhasználhatóságot tesz lehetővé. A motor beépített komponensei között megtalálható a **fizikai rendszer**, az **animációkezelés**, a **vizuális shader -rendszer**, valamint a **felhasználói felületek kialakítását** segítő eszköztár. A fejlesztők több szkriptnyelv közül választhatnak, így a GDScript mellett **C# és C++** is használható, ami megkönnyíti a különböző fejlesztési igényekhez való alkalmazkodást.

A Godot Engine előnyei közé tartozik, hogy **teljesen platformfüggetlen**, és támogatja a játékok exportálását **Windows, Linux, macOS, Android, iOS és Web (HTML5)** rendszerekre. A motor jól integrálható külső eszközökkel, például **Blender -rel, REST API-okkal vagy adatbázisokkal**, továbbá lehetőséget nyújt **VR- és AR-fejlesztésekre**, valamint **többjátékos rendszerek** kialakítására is.

A Godot választása azért indokolt, mert egy **ingyenesen elérhető, átlátható és professzionális fejlesztőkörnyezetet** kínál, amely magas fokú rugalmasságot biztosít. A hivatalos dokumentáció részletes útmutatókat és példákat tartalmaz, míg az aktív fejlesztői közösség fórumokon, blogokon és közösségi platformokon keresztül folyamatosan bővíti a tudásbázist. A motor nyílt forráskódjának köszönhetően a fejlesztők teljes kontrollt gyakorolhatnak a projekt felett, ami különösen fontos a kutatás-fejlesztési és oktatási célú felhasználás során.

1.1.2. A GDScript programozási nyelv

A GDScript a Godot Engine saját, magas szintű programozási nyelve, amelyet kifejezetten a játékfejlesztés megkönnyítésére terveztek. Szintaxisa a Python nyelvre emlékeztet, így könnyen elsajátítható, ugyanakkor teljes mértékben a Godot architektúrájához igazodik. A GDScript szorosan integrált a motor komponenseivel, ezért közvetlenül hozzáfér a jelenetekhez, csomópontokhoz és a motor funkcióihoz. A nyelv dinamikusan típusos, de támogatja az opcionális típusmegadást is, ami növeli a kód biztonságát és átláthatóságát. A GDScript használata lehetővé teszi a gyors prototípus-készítést, ugyanakkor alkalmas komplex logikai rendszerek megvalósítására is. Mivel közvetlenül a motorhoz készült, futása optimalizált és hatékony, így ideális választás a Godot-alapú fejlesztésekhez [2].

1.1.3. A jelenet- és csomópont alapú architektúra

A Godot Engine központi koncepciója a jelenet- és csomópontalapú rendszer, amely a fejlesztés modularitását és rugalmasságát biztosítja. Minden elem — legyen az egy karakter, kamera vagy fizikai objektum — egy csomópont (Node) formájában jelenik meg, amely hierarchikus struktúrában szerveződik. Ezekből a csomópontokból épülnek fel a jelenetek (Scenes), amelyek önálló egységként kezelhetők, majd más jelenetekbe ágyazhatók. Ez a megközelítés megkönnyíti a komponensek újra felhasználását, a projektek áttekinthetőségét és a csapatmunkát. A hierarchikus modell segítségével a fejlesztők hatékonyan tudják kezelni a játéklogikát, az animációkat és a felhasználói felületet egyaránt [3].

1.1.4. A fizikai rendszer és animációkezelés

A Godot beépített fizikai motorral rendelkezik, amely támogatja a 2D és 3D objektumok mozgását, ütközését és dinamikus kölcsönhatásait. A motor külön kezeli a 2D és 3D fizikai rendszereket, így mindkettő optimalizált teljesítményt nyújt. Az animációrendszer szintén fejlett: kulcsképkockás, görbékkel vezérelt, valamint csontvázalapú animációkat is kezel. Az AnimationPlayer és AnimationTree komponensek lehetővé teszik a komplex mozgások és átmenetek precíz irányítását. A Godot vizuális szerkesztője révén az animációk idővonal-alapon hozhatók létre, így a fejlesztés folyamata intuitív és vizuálisan jól követhető [2].

1.1.5. Exportálás és platformfüggetlenség

A Godot Engine egyik legnagyobb előnye a platformfüggetlenség. A kész játékok egyszerűen exportálhatók több operációs rendszerre, beleértve a Windows, Linux, macOS, Android, iOS és Web (HTML5) környezeteket. Az exportálás folyamata automatizált és a beállítások minden

platformhoz külön testre szabhatok. A Godot támogatja a Progresszív Webalkalmazások (PWA) és a mobilalkalmazások fejlesztését is, valamint integrálható külső szolgáltatásokkal, például adatbázisokkal, REST API-okkal vagy felhőalapú megoldásokkal. Ez a rugalmasság lehetővé teszi, hogy a fejlesztők egyetlen forráskódból több eszközre is hatékonyan publikáljanak [2, 14].

1.2. Fejlesztői környezet és eszközök

A játék fejlesztéséhez két fő szoftveres eszközt alkalmaztam: a Godot Engine-t és a Pixelorama pixelgrafikai szerkesztőt. A Godot Engine a fejlesztés központi eleme, amely a játék logikájának implementálását, a jelenetek kezelését, a fizikai rendszert, az animációk kezelését, valamint a felhasználói felület kialakítását teszi lehetővé. A Godot integrált fejlesztői környezete (IDE) lehetővé teszi a kód és a vizuális elemek egyidejű kezelését, valamint a játékok egyszerű exportálását különböző platformokra. A motor beépített eszközei — például a SceneTree, az AnimationPlayer és a TileMap komponensek — lehetővé teszik a moduláris felépítést és az objektumok hierarchikus szervezését, ami különösen fontos egy összetett játékmenet implementálásánál.

A vizuális elemek előállításához a Pixelorama programot használtam, amely egy nyílt forráskódú, 2D pixelgrafikai szerkesztő. A program lehetővé teszi sprite-ok, háttérképek és animációs képkockák készítését, amelyek közvetlenül importálhatók a Godot Engine-be. A Pixelorama egyszerű, intuitív felülete és a rétegkezelés támogatása megkönnyíti a játék vizuális elemeinek szerkesztését, míg az exportált képek formátuma kompatibilis a Godot által használt erőforrás kezeléssel [5].

A két eszköz kombinációja lehetővé teszi a fejlesztés teljes folyamatának lefedését a vizuális tervezéstől a kód alapú logikáig, miközben a nyílt forráskódú jellegük biztosítja a szabad felhasználást és testreszabhatóságot. A fejlesztői környezet kiválasztása során kiemelten fontos szempont volt a könnyű integráció, a dokumentáció és a közösségi támogatás elérhetősége, amelyek a projekt hatékony megvalósítását segítik.

1.3. A 2D játékfejlesztés sajátosságai

A 2D játékfejlesztés során a játékterek, karakterek és interakciók kétdimenziós síkon valósulnak meg, ami jelentősen befolyásolja a játékmechanika és a programozás felépítését. A játék világát sprite -ok és tilemap -ek segítségével lehet modellezni, amelyek egységes koordinátarendszerben helyezkednek el, és a kamera mozgása határozza meg a játékos látóterét.

A tilemap -ok alkalmazása lehetővé teszi a pályák moduláris felépítését, ami elősegíti a pályaszerkesztés hatékonyságát és az újra felhasználhatóságot.

A 2D-s játékok esetében kiemelten fontos a rétegkezelés (z-index), amely meghatározza az objektumok egymáshoz viszonyított láthatóságát, valamint az ütközésdetektálás precíz implementálása. A fizikai kölcsönhatások — például a gravitáció, az akadályokkal való ütközés és a karakter mozgása — a 2D-s sík sajátosságaihoz igazodnak, így optimalizált algoritmusokkal és beépített fizikai komponensekkel valósíthatók meg.

A 2D játékfejlesztés további jellemzője a sprite -alapú animációk kezelése, amelyek kulcsképkockák és animációs ciklusok formájában kerülnek implementálásra. A karakterek mozgása, az ellenségek viselkedése, valamint a vizuális effektek szoros kapcsolatban állnak a játék logikájával, ami különösen fontos egy souls-like jellegű platformer esetében, ahol a pontosság és a játékos interakcióinak visszajelzése alapvető élményelem.

Összességében a 2D játékfejlesztés sajátosságai megkövetelik a vizuális és a logikai komponensek szoros integrációját, valamint a játékmechanikák precíz, koordinátarendszer-alapú megvalósítását. A Godot Engine és a Pixelorama eszközkombinációja lehetővé teszi ezen követelmények hatékony teljesítését, biztosítva a játék funkcionális és vizuális konzisztenciáját [1, 5, 6].

1.4. A souls-like műfaj technikai jellemzői

A „souls-like” műfaj olyan játékmeneti elemeket foglal magában, amelyek a Dark Souls sorozatból váltak ismertté. Ezek közé tartozik a nagy nehézségi szint, a precíz időzítést igénylő harcrendszer, valamint a fokozatosan felfedezhető, összefüggő pályarendszer. A technikai megvalósítás szempontjából ezek az elemek jelentős kihívást jelentenek, különösen egy 2D platformer játék esetében [7].

A harcrendszer megvalósításához fontos a pontosan definiált ütközési mezők (hitboxok) és a karakterek mozgásának finomhangolása. A támadások, védekezések és mozdulatok időzítése kritikus szerepet játszik a játék élményében, ezért a mozgások és animációk szoros szinkronban kell, hogy legyenek a játék logikájával. A támadás- és védekezésmechanizmusok implementálása során a fejlesztőnek figyelembe kell vennie a frame-alapú időzítést és az esetleges invincibility frame-ek kezelését, amelyek a játékos és az ellenségek interakcióit szabályozzák.

Az ellenségek viselkedése és mesterséges intelligenciája (AI) a souls-like játékok kulcseleme. Az AI lehetővé teszi az ellenségek adaptív és kihívást jelentő reagálását, például a támadások előrejelzését, a távolságtartást, vagy a pályaelemekhez való alkalmazkodást. A 2D környezetben az AI implementálása magában foglalja a mozgás útvonalak definiálását, a látómező és a közelharci távolság figyelembevételét, valamint a különböző támadásminták kialakítását.

A „souls-like" játékok további jellegzetessége a checkpoint- és mentési rendszer, amely a játékos halálát követően biztosítja a folytonosságot, ugyanakkor megtartja a kihívás élményét. Emellett a pályák összefüggő struktúrája és a titkos útvonalak, visszatérő helyszínek kialakítása kiemelt figyelmet igényel a játéktér modellezésében.

2. SPECIFIKÁCIÓ

A következő fejezet célja a fejlesztett játék részletes specifikációjának bemutatása. Ebben a részben kerülnek ismertetésre a játék alapvető céljai, mechanikái, felépítése, valamint a működését meghatározó technikai és tervezési döntések. A fejezet kitér a játékmenet fő elemeire, a vizuális és interaktív komponensekre, illetve arra, hogyan valósulnak meg ezek a Godot Engine környezetében.

A specifikáció részletes leírása alapot biztosít a későbbi implementációs folyamat megértéséhez, valamint meghatározza azokat a követelményeket és jellemzőket, amelyek mentén a játék fejlesztése és tesztelése megvalósult.

2.1. Játékmenet

A fejlesztett játék egy kétdimenziós, oldalnézetes souls-like platformer, amely egy nagy, összefüggő, nyitott pályán játszódik. A játékos szabadon felfedezheti a világ különböző területeit, miközben ellenségekkel küzd, új útvonalakat tár fel, és fokozatosan fejlődik. A játékmenet központi eleme az új helyszínek és titkos átjárók felfedezése, amelyek gyakran új ellenségeket, kihívásokat és jutalmakat rejtenek. A felfedezés folyamatos jutalmazása fenntartja a játékos motivációját, és lehetőséget ad a saját útvonal megválasztására.

A játék során a játékos egyetlen, univerzális erőforrást gyűjt, amelyet minden fontos tevékenységhez felhasználhat. Ezzel az erőforrással lehet fejleszteni a karakter tulajdonságait, új felszereléseket vagy képességeket megszerezni, illetve bizonyos interaktív elemeket aktiválni. Az egységes erőforrásrendszer tudatos döntés elé állítja a játékost: kockáztasson és

tovább haladjon több gyűjtött egységgel, vagy használja fel azokat biztonságos pontokon a fejlődés érdekében. Ez a mechanika a souls-like játékokra jellemző kockázat-jutalom elv egyik legfontosabb eleme.

A harcrendszer a műfaj sajátosságait követi: minden támadásnak, ugrásnak és védekezésnek súlya és időzítése van. A játékosnak fel kell ismernie az ellenségek mozgásmintáit, és ezek alapján kell reagálnia. A halál a játékmenet szerves része — a játékos ilyenkor visszakerül a legutóbbi aktivált ellenőrzőpontra, és elveszíti a nála lévő erőforrásokat, de lehetőséget kap azok visszaszerzésére. Ez a ciklikus előrehaladás és újra próbálás adja a játék egyik legfontosabb ritmusát és feszültségét.

2.2. Felhasználói felület

A játék felhasználói felülete (UI) minimalista, mégis funkcionálisan gazdag kialakítást követ, amely a *souls-like* játékokra jellemző letisztult vizuális stílust idézi. A cél a játékos teljes bevonása a játékmenetbe, ezért a UI elemek nem tolakodóak, kizárólag a legszükségesebb információkat jelenítik meg. A felhasználói felületet több, különböző funkciót betöltő menürendszer alkotja, amelyek mind a Godot engine saját felületkezelő komponenseire épülnek [12].

2.2.1. Kezdőképernyő

A játék indításakor a játékos a kezdőképernyőre kerül, ahol elérhető a legfontosabb főmenü-elemek: **Új játék**, **Betöltés**, **Beállítások** és **Kilépés**. A háttérben egy statikus, atmoszférikus háttérkép vagy animált jelenet fut, amely illeszkedik a játék hangulatához.
A *Load* (betöltés) menüben megjelennek a korábban mentett állások, vizuális előnézettel (pl. karakter szintje, helyszín neve, mentési időpont). A mentési rendszer manuális mentést nem engedélyez a játékon belül, kizárólag a pihenőpontok (*checkpoints*) aktiválásakor történik automatikus mentés.

2.2.2. Játékon belüli főmenü

A játékon belül a főmenü, innen a játékos elérheti a **Beállítások**, **Főmenübe vissza**, **Kilépés**, valamint a **Fejlődés** és **Barkácsolás** menüket. A menü navigációja egyszerű, egységesen a billentyűzet és kontroller támogatását is biztosítja, a Godot Control csomópontjain alapuló hierarchikus felépítéssel.

2.2.3. Beállítások menü

A beállítások menü a játékos számára lehetővé teszi az alapvető rendszerbeállítások módosítását, mint a **hangerő**, **felbontás**, valamint a **teljes képernyős mód**. A felület logikusan strukturált, a változtatások azonnal alkalmazhatók, és a beállításokat a játék automatikusan elmenti.

2.2.4. Fejlesztési menü

A fejlődés menü közvetlenül a *souls-like* játékok karakterfejlesztési rendszerét idézi. A játékos itt költheti el a gyűjtött erőforrásait különböző attribútumokra, mint például **életerő**, **állóképesség**, **sebzés** vagy **gyorsaság**. A fejlesztés kizárólag az *ellenőrző pontoknál* lehetséges, ezzel ösztönözve a taktikus döntéshozatalt.

A menü vizuálisan a klasszikus *Dark Souls* stílust követi: sötét tónusú háttér, finom tipográfia, és letisztult statisztika-megjelenítés jellemzi. Az új értékek kijelölés után csak megerősítésre kerülnek véglegesen alkalmazásra, így elkerülhető a véletlen fejlesztés.

2.2.5. Barkácsolás menü

A barkácsolás rendszer lehetővé teszi különböző tárgyak, fegyverek vagy fejlesztések készítését a játék során gyűjtött alapanyagok felhasználásával. A menü átlátható kategóriákba rendezi a recepteket, és valós időben mutatja, mely összetevők állnak rendelkezésre.

A fejlesztés és készítés folyamata a gyűjtött univerzális erőforráshoz kötődik, ami szoros kapcsolatot teremt a játékmenet és a gazdasági rendszer között. A játékos így minden döntésénél mérlegelni kényszerül: új tárgyat készít, vagy inkább karaktert fejleszt.

2.2.6. Ellenőrző pont menü

Az ellenőrző pont menü a játék egyik központi eleme, amely a pihenés és mentés funkcióját látja el. A játékos itt biztonságosan pihenhet, visszatöltheti életerejét és állóképességét, valamint fejlesztheti karakterét vagy készíthet tárgyakat is bizonyos esetekben.

Az ellenőrző pont aktiválásával a játék automatikusan ment, azonban minden normál ellenség újraéled, hasonlóan a *Dark Souls* játékok rendszeréhez. Ezzel a megoldással a játék megőrzi a feszültséget, miközben a fejlődés biztonságos pontjait is biztosítja.

2.2.7. HUD

A HUD a játékos számára egyik legfontosabb felület, aminek minimálisnak kell lennie, és minden szükséges információt közölnie kell a játékossal, többek közt interakciók, életerő fontos gyűjthető tárgyak, jelenlegi fegyver/képesség jelzése, olykor szituációtól függően változhat, akár a teljes felület. Azért is fontos mert a játékos ezt a felületet látja a legtöbbször.

2.3. Játék rendszerek

2.3.1. Harcrendszer

A harcrendszer a játék egyik központi eleme, amely a souls-like műfaj alapvető jellemzőit követi: a mozdulatok időzítése, a támadások súlya, valamint az ellenségek viselkedésmintáinak felismerése kulcsfontosságú [8].

A karakter különböző támadásokat hajthat végre, amelyek stamina felhasználásával járnak. Amennyiben a stamina teljesen kimerül, a karakter nem képes további támadásra vagy kitérésre, így a játékosnak tudatosan kell gazdálkodnia az erőforrásaival.

A találatok mindkét félre nézve jelentős hatással bírnak – a sebzés mértéke, a támadási sebesség és a karakter mozgása mind kiegyensúlyozott arányban lettek megtervezve, hogy a játék dinamikus, de kihívásokkal teli legyen.

2.3.2. Életerő- és állóképesség-rendszer

A képernyő felső részén helyezkedik el a **HUD**, amely tartalmazza a **health** és **stamina** sávokat, valamint a gyűjtött erőforrás mennyiségét jelző számlálót, amely dinamikusan jelenik meg, vagyis csak akkor látszódik, ha az értéke változik, vagy ha bizonyos menü elemek indokolják pl.: barkácsolás menü.

Az életerő csökken ellenséges támadások, csapdák vagy környezeti veszélyek hatására. A **stamina** minden akcióval (támadás, ugrás, kitérés) csökken, és idővel automatikusan regenerálódik, amennyiben a játékos nem hajt végre újabb akciót.

Ez a két sáv biztosítja a játékmenet dinamikus ritmusát, és közvetlenül befolyásolja a játékos taktikai döntéseit a harc közben. A **stamina** gyors regenerációt tükröz, viszont a **health** nem, vagy csak lassan, esetleg bizonyos szintig regenerálódik.

2.3.3. Eszköztár rendszer

A játék eszköztár rendszere a tárgyak, fegyverek és egyéb gyűjthető elemek kezelésére szolgál. A játékos a begyűjtött tárgyakat kategóriák szerint rendezve láthatja, és a különböző felszereléseket közvetlenül innen tudja aktiválni. A rendszer támogatja a felszerelhető tárgyak kezelését, valamint a különböző fogyóeszközök (pl. gyógyitalok, erősítők) gyorselérését.

Az eszköztár struktúrája moduláris, a Godot engine Dictionary, Array, és Resource adatszerkezeteire épül, ami lehetővé teszi a dinamikus bővítést és az egyedi tárgyak paramétereinek tárolását [4, 15].

2.3.4. Gyorselérési kerék rendszer

A HUD -on elérhető a gyorselérési kerék a játékos számára lehetővé teszi, hogy a legfontosabb tárgyait gyorsan elérje harc közben, a Dark Souls játékokhoz hasonló módon.

A kerék forgatható, és egyidejűleg korlátozott számú (például négy) tárgy vagy eszköz helyezhető el rajta. A kiválasztott elem a képernyőn vizuálisan kiemelt formában jelenik meg. Ez a megoldás egyszerre biztosít gyors hozzáférést és taktikai korlátokat, hiszen a játékosnak előre el kell döntenie, mely tárgyakat szeretné magával vinni a csatába.

2.3.5. Erőforrás rendszer

A játékban található univerzális erőforrás minden előrehaladás kulcsa. Ez a gyűjthető érték jutalmazza az ellenfelek legyőzését és a felfedezést.

A megszerzett egységek száma kijelzésre kerül a HUD-on. Az erőforrást a játékos többféle módon használhatja: karakterfejlesztésre, tárgykészítésre (crafting), vagy bizonyos környezeti interakciók aktiválására.

Halál esetén az erőforrás ideiglenesen elveszik, azonban visszaszerezhető a halál helyszínére visszatérve – ez a mechanika közvetlenül a souls-like játékok jól ismert kockázat-jutalom struktúráját valósítja meg.

2.3.6. Ellenőrzőpont rendszer

Az ellenőrzőpontok stratégiailag elhelyezett pontok a pályán, amelyek lehetőséget nyújtanak a pihenésre, fejlődésre és mentésre. Az ellenőrzőpont aktiválásakor a játék automatikusan menti az állást, feltölti a játékos életerejét és állóképességét, valamint visszaállítja az összes legyőzött ellenséget (a főellenségek kivételével). A mentés megvalósításának egyszerűnek kell lennie, de védettnek, vagyis alapvetően automatikus mentésekkel kell elvégezni.

Ez a megoldás biztosítja a játék folyamatos kihívását, miközben a játékos fejlődése és előrehaladása nem veszít értelmet. A rendszer logikája a Dark Souls táborhelyeihez (bonfire) hasonlóan működik.

3. TERVEZÉS

A Tervezési fázisban kerültek elkészítésre a karakter, ellenség, főellenség és a pálya tervei, a játék tervezésnél fontos szempont volt, hogy idézzük a régi 8-bites színmélységű videójátékokat ezáltal egy „retro” stílust kölcsönözzünk a megjelenést illetően. Annak érdekében, hogy elérjük a *souls-like* stílust az interfészekhez kapcsolódóan, utána kellett járjak, hogy az egyes felhasználói felületek hogyan jelennek meg hasonló játékokban szerencsére van egy interneten elérhető elég nagy gyűjtemény, amely tele van videójátékok felhasználó felületeinek felépítésével [11, 12].

3.1. Karakter- és ellenségtervezés

A játék vizuális koncepciójának egyik központi eleme a **retro, pixeles megjelenés**, amelyhez szándékosan egységesített, minimalista animációk kerültek kialakításra. A karakter és az ellenségek animációi **egy közös *spritesheet -ben*** kaptak helyet, amely egyszerre támogatja a nosztalgikus esztétikát és a hatékony technikai megvalósítást. A közös *spritesheet* használatának több előnye is van a tervezési és fejlesztési folyamat során [13]:

- **Egységes vizuális stílus:** mivel minden sprite ugyanazon fájlban található, biztosítható, hogy az animációk ugyanazzal a színpalettával, méretaránnyal és stílusjegyekkel készüljenek. Ez különösen fontos a retro megjelenés szempontjából, ahol az összhang sokat számít.

- **Könnyebb iteráció és módosítás:** a fejlesztés során folyamatosan változnak az animációk, ezért egyetlen fájl kezelése egyszerűbb, mint több különálló asset karbantartása. A Pixelorama használata mellett a spritesheet frissítése gyors és átlátható marad.

- **Teljesítményoptimalizálás:** a Godot engine hatékonyabban kezeli az olyan projekteket, ahol kevesebb külön képfájlt kell betölteni, így csökken a memóriahasználat, és kevesebb draw call jön létre, ami növeli a teljesítményt [17].

A *spritesheet* -alapú munkamenet tehát nem csupán technikai döntés volt, hanem a játék hangulatát és a fejlesztés hatékonyságát egyaránt támogatta. Az egységes forrásfájl lehetővé tette, hogy a karakter és az ellenségek vizuálisan és animációs logikájukban is koherens rendszert alkossanak.

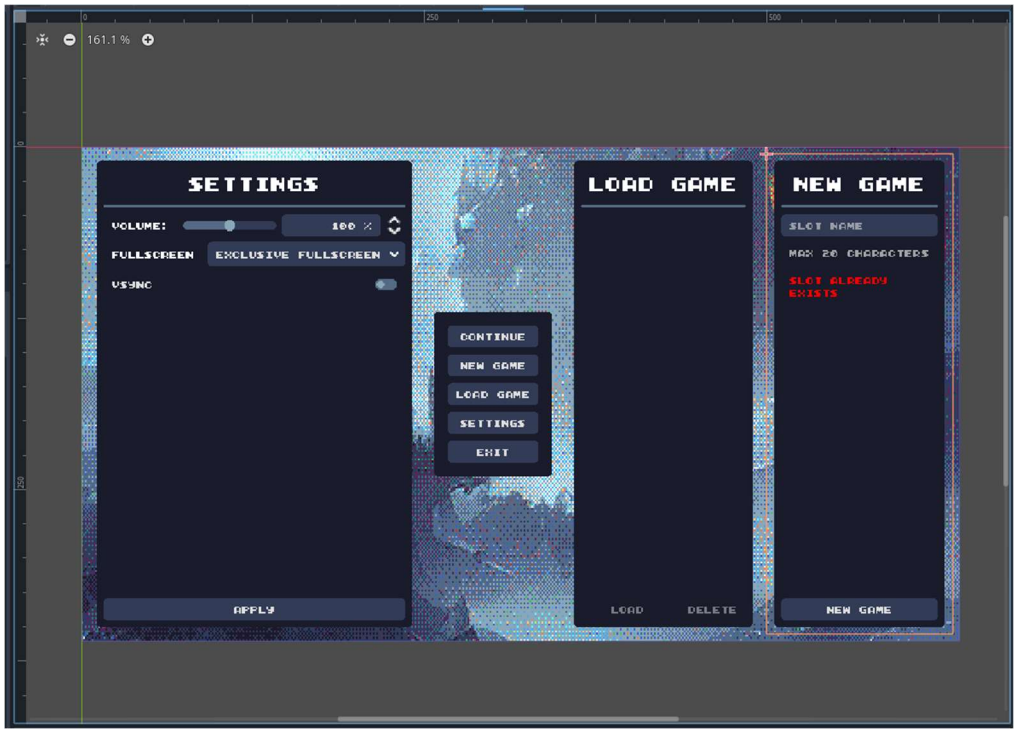
3.2. Felhasználói felület

3.2.1. Főmenü

A főmenü tervezésénél fontos, hogy a következő opciók elérhetőek legyenek (3.1 ábra):

- Játék folytatása
- Új játék (3.1 ábra)
- Játék betöltése
- Beállítások (3.1 ábra)
- Kilépés

Ezek az opciók majd almenüket jelenítenek meg ahol a hozzájuk tartozó műveletek elvégezhetőek például a betöltés almenüben lehet a mentéseket betölteni, vagy letörölni. Mivel egy pixel art játékról van szó, ezért nem sok beállítás áll rendelkezésünkre, de itt különböző beviteli mezők segítségével módosíthatjuk a játék különböző beállításait. A játék folytatása a legutóbbi mentett állast tölti be, a kilépés pedig kilép a programból



3.1 ábra – Főmenü

3.2.2. Játékon belüli Főmenü

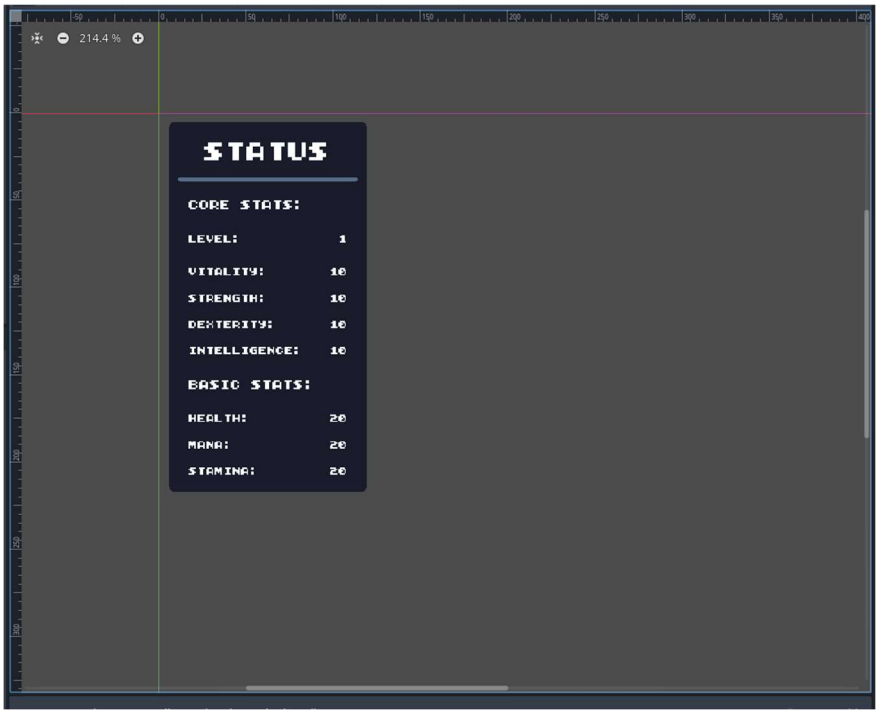
Ez az interfész hasonló módon viselkedik, mint a főmenü. A főmenüben lévő opciókhoz ad néhány másik egy már megkezdett játékhoz szükséges opciót, az interfész nem állítja meg a játék futását, de a karakter inputokat blokkolja.

3.2.3. Betöltés menü

A játék állás betöltésére és letörlésére használatos felület, ez egy almenü, ami jelen van a főmenüben és a játékon belüli főmenüben is. Egy egyszerű lista, ami a mentéseinket listázza, egy listaelem egy időpontból egy esetleges képből áll, ami tükrözi a játékmenet állását. (3.1 ábra)

3.2.4. Karakter menü

A karakter menü arra szolgál, hogy megtekintsük a karakter jelenlegi státuszát életerő, kitartás, szint… stb. (3.2 ábra)



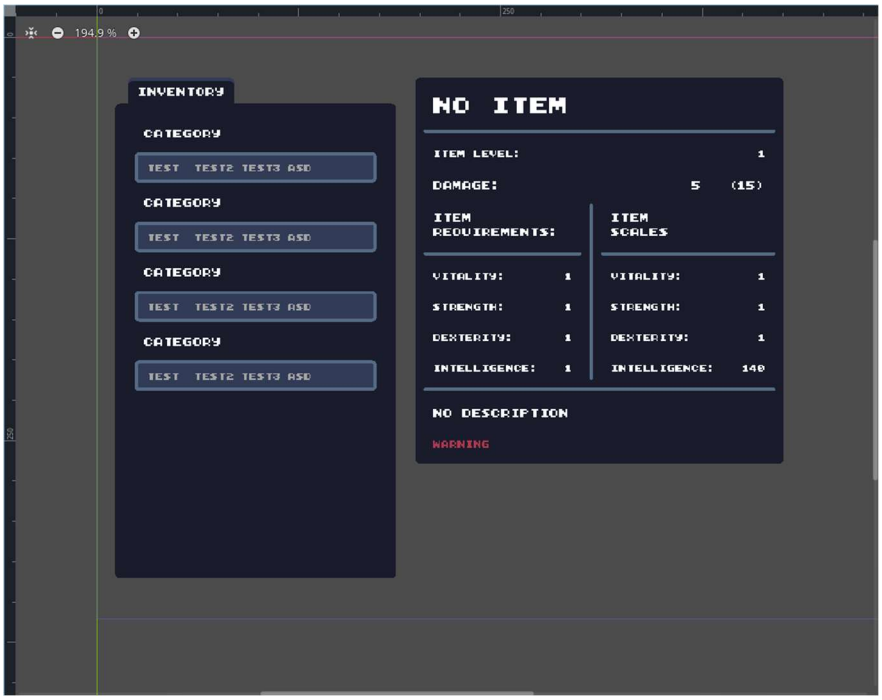
3.2 ábra – Karakter státusz menü

3.2.6. Karakter szintlépés menü

A játékos ebben a menüben tudja a karaktere szintjét növelni, ez a menü nagyban hasonlít a Dark Souls, illetve Elden Ring nevű játékok karakter fejlesztési menüjére. (3.4 ábra)

3.2.7. Karakter leltár menü

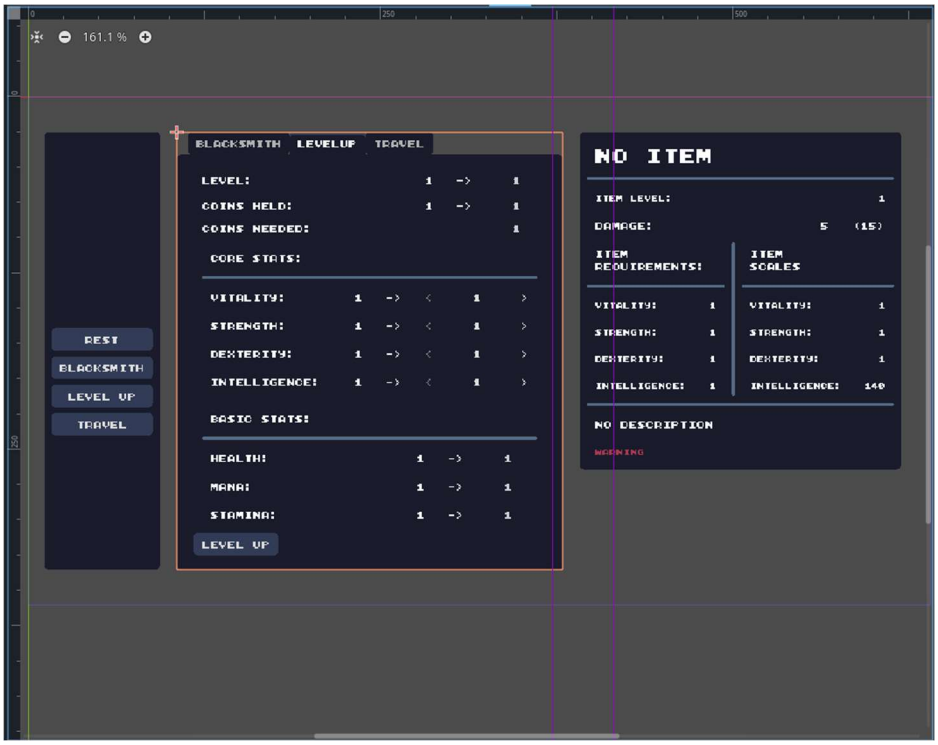
A karakter leltárban megnézhetjük a különböző már felvett tárgyainkat, illetve beállíthatjuk, hogy a gyorselérési interfészen milyen tárgyak legyenek, bármilyen tárgyat elhelyezhetünk oda.



3.3 ábra – Karakter szintlépés menü

3.2.8. Ellenőrző pont menü

Ez a menü nagyon fontos része a játéknak mivel itt érhető el a karakter fejlesztési almenü, barkácsolás almenü, illetve gyorsutazás almenü, és itt lehet a játékot elmenteni úgy, hogy a karakterünk „pihen” egyet, amivel továbbá visszaállítunk minden ellenséges egységet, kivéve a főellenségeket. (3.4 ábra)



3.4 ábra – Karakter szintlépés menü

3.2.9. Barkácsolás menü

A barkácsolás menü arra szolgál, hogy a karakterünk bizonyos felvett tárgyakból tudjon készíteni más tárgyakat, például kardok, kalapácsok és hasonlók. (3.5 ábra)

3.5 ábra – Barkácsolás menü

3.2.10. Gyorsutazás menü

Ez a menü egy egyszerű lista a már felfedezett ellenőrzési pontokról, minden pontnak van egy különleges neve ezáltal segítve a játékos a tájékozódásban. (3.6 ábra)

3.6 ábra – Gyorsutazás menü

3.3. Pálya tervezés

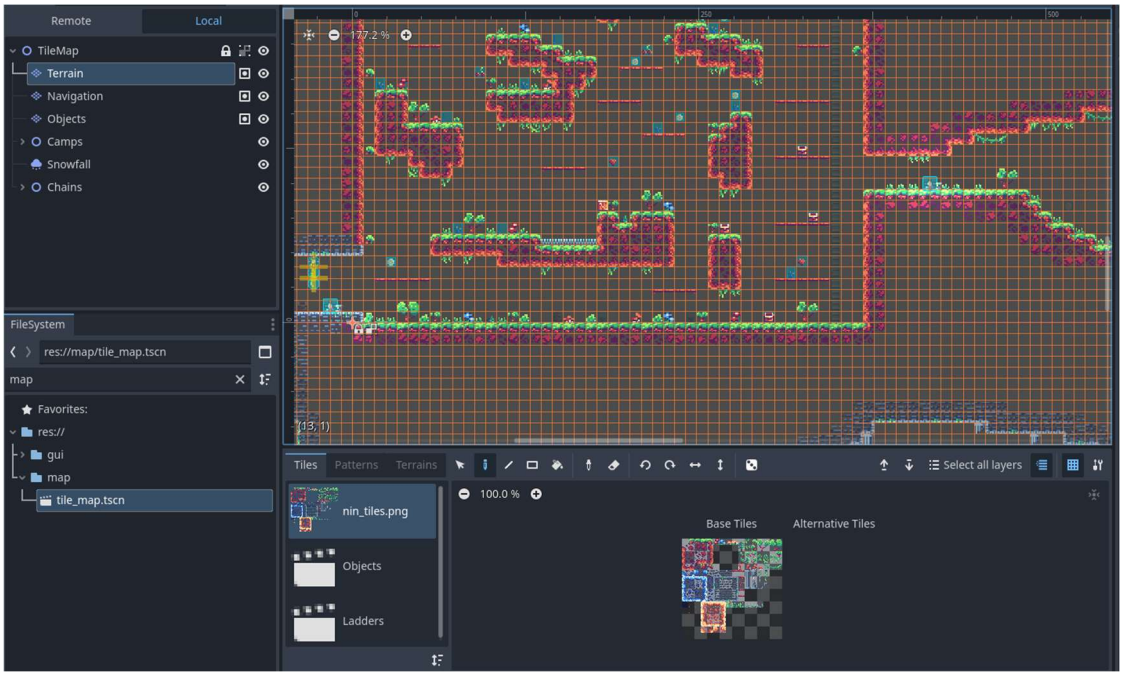
A játék pályájának megtervezése kulcsfontosságú eleme a fejlesztési folyamatnak, hiszen ez határozza meg a játékos élményének ritmusát, a felfedezés érzését, valamint a kihívás fokozatos növekedését. (3.7 ábra)

A pályatervezés során a cél egy olyan összefüggő, nyitott felépítésű világ létrehozása, amely szabad felfedezést biztosít, ugyanakkor logikusan kapcsolódó területeket tartalmaz. A pálya elrendezése a „souls-like” stílushoz hűen több rövidítést, visszavezető útvonalat és rejtett területet tartalmaz, amelyek a felfedezést és a tanulást jutalmazzák.

A pálya megtervezéséhez a Godot engine -be beépített „*TileMap*” csomópont tökéletes, akár az elő pályát elkészítéséhez is használható, sőt ajánlott is. Festésre hasonlító módon lehet a pályát befolyásolni vele, nagyon hasznos, hogyha sokszor iterálunk a pályán vagy akár, ha csak tervezésre használjuk. A használatához szükséges egy csempe elrendezésű képfájl szükséges, ezután lehetőségünk van mintákat, létrehozni, hogy elősegítsük munkánkat.

A pálya tervezésnél figyelembe kell venni hova helyezünk le ellenőrző pontokat, ládákat, csapdákat, ellenségeket, és nemutolsó sorban főellenségeket, ugyanis ezeken múlik a játék nehézségi görbéje, és a játékmenet dinamikája.

Mivel egy nagy nyitott világunk van ezért gondosan kell a pályát megtervezni és lehetőség szerint figyelni arra hogyha a jövőben bővítésre szorulna akkor, ne legyen szükség arra, hogy a már meglévő pályát módosítsuk.



3.7 ábra – Pálya részlet

3.4. Rendszerek megtervezése

A rendszerek megtervezése során nagyban támaszkodtam a Godot engine hivatalos dokumentációjára, amely részletes útmutatást ad különböző programozási minták (design pattern -ek) implementálásához. Az motor rugalmasságának köszönhetően a legtöbb funkció saját kóddal, de jól strukturált módon megvalósítható, így a projekt skálázható és könnyen karbantartható marad.

A mentési rendszer kezeléséhez a *Singleton* (egyke) tervezési mintát alkalmaztam. Ennek lényege, hogy létrehozok egy globális „Mentés” osztályt, amelyből csak egyetlen példány létezik a futásidő során. Ez az objektum felelős az adatok – például a játékos pozíciója, statisztikái, illetve a pálya aktuális állapota – mentéséért és betöltéséért. A *Singleton* globális elérhetősége lehetővé teszi, hogy a mentési funkciót a játék bármely pontjáról könnyedén meghívjam, ezzel csökkentve a kódismétlést és növelve az átláthatóságot.

Az ellenségek viselkedését és a karakter logikáját egyaránt a *State Machine* (állapotgép) mintára építettem. Ez a megoldás különösen hasznos olyan helyzetekben, ahol egy objektum több, egymástól jól elkülöníthető állapotban lehet (például: „járőrözik”, „támad”,

„visszavonul”, „meghalt”). Az állapotgép használata biztosítja a kód modularitását és egyszerű bővíthetőségét: új viselkedési minták könnyen hozzáadhatók anélkül, hogy a meglévő logikát módosítani kellene.

A rendszertervezés során továbbá külön figyelmet kapott a harcrendszer, a tárgykezelés (*inventory*) és a fejlődési mechanizmusok alapjainak előkészítése. Ezek mind önálló alrendszerekként működnek, de szorosan integrálódnak a fő játékmenetbe. A cél az volt, hogy minden funkció önállóan is tesztelhető legyen, ugyanakkor egymással zökkenőmentesen kommunikáljanak.

3.4.1. Mentés rendszer

A mentésrendszer tervezésénél elsődleges cél volt a megbízhatóság és a játékfolyamat integritásának megőrzése. A rendszer nem hagyományos, felhasználó által kezelt mentéseket alkalmaz, hanem automatikus mentési mechanizmust, amely minden fontos eseménynél (például pihenőpontnál vagy halál után) frissíti az aktuális állapotot. A tervezés lényege, hogy a játék mindig csak a legutóbbi, nem korrupt mentést töltse be, így elkerülhetők a sérült fájlokból vagy megszakított folyamatokból adódó hibák. A tervezés során fontos volt, hogy a mentési adatokat logikailag elkülönítve, de egységes formátumban kezelje a rendszer. Minden mentés meta adatokat is tartalmaz, mint például a játékidő, a pálya neve, a karakter szintje és az utolsó mentés időpontja. Ezek segítségével a mentések gyorsan azonosíthatók a felhasználói felületen, illetve megkönnyítik a hibakezelést is.

A rendszer továbbá úgy lett megtervezve, hogy támogassa a biztonságos mentést és visszatöltést, azaz elkerülje a megsérült fájlokból adódó problémákat. Ezt a tervezés szintjén többek között azzal lehet biztosítani, hogy a mentés két fázisban történik: először ideiglenes fájlba kerülnek az adatok, majd csak sikeres írás után cseréli le a rendszer az előző mentést.

A játékmechanikához igazodva a mentési folyamat nem manuálisan, hanem az ellenőrzőpont-rendszeren keresztül történik, ezzel megőrizve a souls-like játékokra jellemző kockázat–jutalom egyensúlyt. A tervezés célja az volt, hogy a mentési pontok elhelyezése és működése dinamikusan illeszkedjen a pálya felépítéséhez, valamint a játékos előrehaladásának ritmusához.

3.4.2. Állapot rendszer

Az állapotgép (State Machine) rendszer tervezésének lényege, hogy a játékban szereplő karakter – legyen az a játékos vagy egy ellenség – több, egymástól jól elkülöníthető állapotban

is lehet, mint például „áll”, „mozog”, „támad”, „sebződik” vagy „meghalt”. Az ilyen jellegű viselkedések kezelésére a Godot engine alapvetően csak az animációk szintjén biztosít állapotkezelést, azonban a komplex logikai működéshez – például mozgás, támadások vagy események kombinációjához – egy saját, általános célú állapotgép-rendszer megvalósítása szükséges [8, 10].

A tervezés során cél volt, hogy minden állapot külön, jól elkülöníthető modul legyen, amely saját felelősséggel rendelkezik, és csak a neki releváns logikát tartalmazza. Ez a megközelítés növeli a kód átláthatóságát és csökkenti a hibalehetőségeket. Az állapotok között kizárólag definiált feltételek (triggerek) alapján történhet átmenet, ezzel biztosítva a rendszer stabil működését.

Az állapotgép továbbá egységes keretrendszert biztosít a játékos és az ellenségek viselkedésének leírására, így a fejlesztés során ugyanaz a logikai struktúra újrahasznosítható mindkét esetben. Tervezéskor fontos szempont volt, hogy az állapotok bővíthetők és testre szabhatok legyenek anélkül, hogy a meglévő logikát módosítani kellene – például új támadástípus vagy mozgási minta egyszerűen hozzáadható új állapotként. Ezzel a megközelítéssel a játék karakterei dinamikusabban, kiszámíthatóbb módon viselkednek, miközben a fejlesztési folyamat is strukturáltabbá és átláthatóbbá válik.

3.4.3. Harc rendszer

A harcrendszer tervezésének egyik fő célja az volt, hogy a játék megőrizze a souls-like műfajra jellemző taktikus, időzítésalapú és büntető jellegű küzdelmeket. Ennek megfelelően a rendszer nem pusztán animációk egymásutánját jelenti, hanem egy olyan komplex logikai keretet, amely a játékos és az ellenségek közötti interakciókat pontosan és következetesen kezeli. A tervezés során kiemelt figyelmet kapott a sebzéslogika, az ütközéskezelés, a támadási fázisok és a kitérési mechanizmusok összhangja.

A harcrendszer alapját a hitbox/hurtbox alapú megközelítés képezi, amely lehetővé teszi a támadások precíz ütközésdetektálását. A rendszer külön kezeli a vizuális animációt és a tényleges sebzési területet, így az animációk nincsenek közvetlenül összekötve a logikai sebzéssel. Ez a megközelítés nagyfokú rugalmasságot biztosít, hiszen a támadások időzítése, távolsága és hatásai később is könnyen módosíthatók anélkül, hogy az animációkat újra kellene szerkeszteni.

A védekezés és kitérés szintén kulcsfontosságú elemei a harcrendszernek. A kitérés (dodge/roll) időzítése az ún. i-frame (invulnerability frame) megközelítést használja, vagyis a karakter a mozdulat bizonyos szakaszában sérthetetlen. A tervezés során törekedni kellett az egyensúly megtartására: a kitérés legyen hatékony, de ne túl erős, hogy továbbra is megmaradjon a kihívás és a gondolkodásra épülő játékmenet.

Összességében a harcrendszer tervezése szoros együttműködést igényelt a mozgás-, animáció- és állapotgép-rendszerrel. A cél egy olyan struktúra kialakítása volt, amely egyszerre precíz, kiszámítható és kielégítő játékélményt nyújt, miközben fejlesztői oldalról átlátható, moduláris és könnyen bővíthető marad.

3.4.3. Eszköztár- és tárgyrendszer

Ez a rendszer a játék egyik központi eleme, mivel közvetlenül befolyásolja a játékos fejlődését, harci lehetőségeit és a játékmenet stratégiai mélységét. A rendszer tervezésénél elsődleges szempont volt, hogy a souls-like műfajra jellemző korlátozott erőforráskezelés, valamint a tárgyaknak és felszereléseknek tulajdonított magas jelentőség észrevehető legyen a játék működésében.

A tervezés alapját egy moduláris, adatvezérelt tárgyrendszer képezi, amelyben minden tárgy — legyen az fegyver, páncél, gyógyító eszköz vagy alapanyag — közös, egységes adattáblában definiált tulajdonságokkal rendelkezik. A tárgyak attribútumai magukban foglalhatják többek között a sebzésértékeket, a védelmi értékeket, a használati korlátozásokat, a ritkaságot és az esetleges speciális effektusokat. Ez lehetővé teszi a rendszer könnyű bővítését, mivel új tárgytípusok hozzáadása nem igényli a meglévő kód jelentős módosítását.

Az eszköztár kialakításánál fontos tervezési elv volt az áttekinthetőség és a gyors hozzáférhetőség. A játék a souls-like játékokra jellemző gyorsváltó rendszerrel (gyorselérési kerék) is rendelkezik, amely harc közben is lehetővé teszi bizonyos tárgyak — például gyógyító eszközök vagy dobófegyverek — gyors elérését. Ennek tervezésekor arra kellett ügyelni, hogy a kerékre helyezett tárgyak mindig szinkronban legyenek a fő eszköztárral, így elkerülhető a tárgy többszöröződés vagy az esetleges konzisztencia-problémák.

A tárgyhasználat és a felszerelésváltás tervezésénél a cél az volt, hogy a folyamat következetes, jól strukturált logikai lépésekből álljon. A karakter felszerelésének módosítása például nem csupán vizuális változást jelent, hanem több rendszert is érint: a statisztikák újraszámítását, a hitbox-paraméterek módosítását vagy akár animációs állapotok frissítését.

Emiatt fontos volt egy olyan központi kezelő (inventory manager) megtervezése, amely a különböző rendszerek között biztosítja az adatáramlás egységességét.

A rendszerhez kapcsolódik a barkácsolás mechanika is, amely az összegyűjtött erőforrások felhasználásával teszi lehetővé új tárgyak létrehozását. A tervezés során arra kellett törekedni, hogy a crafting logikája a játék világába illeszkedjen, és egyértelmű előrehaladási útvonalat adjon a játékosnak. Mivel ugyanazt a nyersanyagot használja fel a játékos a karakter fejlesztéséhez és a tárgykészítéshez is, a rendszer olyan módon lett megtervezve, hogy döntési helyzeteket teremtsen, ezzel mélyítve a játékmenetet.

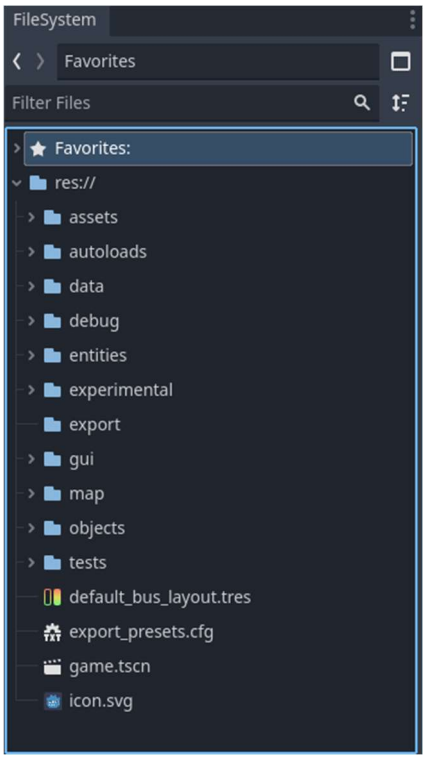
4. MEGVALÓSÍTÁS

4.1. Projektstruktúra és mappafelépítés

A projekt fejlesztése során a mappastruktúra több alkalommal is módosult, mivel a kezdeti fázisban még nem volt egyértelmű, melyik rendszer milyen formában fog elkészülni, illetve hova lesz érdemes elhelyezni. A végleges struktúra kialakításánál azonban az volt a fő elv, hogy nem fájlkiterjesztés, hanem funkcionális összefüggés alapján történjen a források csoportosítása. Ennek megfelelően például a játékoshoz tartozó minden komponens — jelenet, script, kamera, animáció — a /entities/player mappában kapott helyet, így egy logikai egységbe kerültek. A projekt végleges mappastruktúrája a következőképpen alakult:

- /assets (képi, illtetve hanganyagok, továbbá effektek stb…)
- /autoloads (singleton -ok)
- /data (a tárgy rendszernek)
- /debug (FPS számláló és különféle diagnosztikai UI felületek/rendszerek)
- /entities (játékos, ellenfelek)
- /experimental (Még nem elkészült szeparált rendszerek/funkciók)
- /export (Az exportált projekt, nem került verzió kezelésre)
- /gui (A játék UI elemei)
- /map (a térkép, és egyéb hozzákapcsolódó elemek)
- /objects (A térképre helyezett objektumok)

- /tests (A rendszerek tesztjei, és egyéb tesztek)



4.1 ábra – Mappa struktúra

A végleges struktúra célja az volt, hogy könnyű legyen megtalálni egy-egy funkcióhoz tartozó összes komponenst, biztosítsa a rendszer moduláris bővíthetőségét, a fejlesztés során kialakult ismételt tesztelési igényeket külön mappában lehessen kezelni, és egy átlátható, hosszú távon is fenntartható projektet hozzon létre.

4.2. A játékos karakter megvalósítása

4.2.1. Jelenet felépítése

A Godot jelenetalapú rendszerét kihasználva a játékos egy több rétegből felépített Node2D alapú jelenetként készült. A legfontosabb komponensek:

- *CollisionShape2D* / *CharacterBody2D*: a fizikai mozgásért és ütközésért felelős csomópont.
- *AnimatedSprite2D* és *AnimationTree*: a mozgáshoz és harci műveletekhez használt animációk kezelése.
- Hitbox / Hurtbox csomópontok: külön kezelik a támadások által okozott sebzést és a karakter által elszenvedhető találatokat.

- *Camera2D*: a játékoshoz rögzített kamera, amely leköveti a mozgást.
- *State Machine script*: a játékos logikáját állapotokra bontja (idle, walk, run, jump, fall, attack, dash stb.).
- HUD: Grafikus interfészek, például eszköztár, státusz indikátorok stb.

Ez a felosztás lehetővé teszi a moduláris frissítéseket: egy új mozdulat, támadás vagy animáció hozzáadása nem igényli a teljes rendszer átírását. A legfontosabb csomópont a *CharacterBody2D* ez vezérli a fizikai hatások nagy részét, ezzel mozgatjuk a karaktert.

4.2.2. Mozgás és interakciók

A karakter mozgását a Godot *CharacterBody2D* komponense végzi, amely a beépített fizikai motort használja. A mozgás logikája több komponensre oszlik:

- Alapmozgás: séta, futás, irányváltás gyors reakcióval.
- Ugrás és esés: a gravitáció és az ugrási erő paraméterezhető, így a mozgás érzete könnyen finomhangolható.
- Dodge / Roll: souls-like játékoknál meghatározó mozdulat, rövid ideig „invincibility frame” -et biztosít, vagyis ideiglenesen kikapcsoljuk az ütközés detektálást.
- Platformérzékelés: külön ellenőrzések felelnek azért, hogy a karakter mikor van földön és mikor levegőben, illetve falon.
- Falugrás: Ha a karakter a fal mellett közvetlenül tartózkodik akkor az ugrás gomb lenyomásával tud a faltól ellenkező irányba ugrani, így akár falat „mászni” is lehetséges.

A mozgásvezérlés figyelembe veszi a stamina rendszert is: futás, kitérés és bizonyos támadások stamina-fogyasztással járnak. A bizonyos „*One-way* platformok” -ra egy viszonylag egyszerű kezelés van, tulajdonképpen a platform elkészítésekor a platform *Collision* objektumán be kell jelölni a „*One way collision*” opciót, ez azt eredményezi, hogyha a platformot alulról közelítjük meg akkor szimplán át tudunk rajta ugrani, de megtart minket mikor leesnénk, a platformról való lefelé lejutáshoz nincs beépített megoldást, ám ha tudjuk hogyan működnek akkor egyszerűen megoldható a probléma. A megoldás annyi, hogy egy gomb lenyomására 1px -el lejjebb helyezzük a karaktert, és onnan a gravitáció megoldja a többit.

4.2.3. Állapotgép megvalósítás

A játékos minden akciója egy azonosított állapotban valósul meg. A manuális state machine kialakításának előnyei:

- áttekinthető logika
- pontos kontroll a beviteli (input) események felett
- egyszerű bővíthetőség új mozdulatokkal és animációkkal

A legfőbb állapotok:

- Default: állás/mozgás
- Walk / Run: mozgás
- Fall: vertikális mozgások
- Attack: közelharci animációk és sebzéskiváltás
- Cast: távolharci animációk, illetve lövedékek létrehozása
- Dodge / Roll: sérthetetlenségi ablakot tartalmaz
- Climb: Létra mászás, és animációk
- Death: halál animáció, amely után ellenőrző pontra történő visszatöltés következik

Az állapotgép szorosan együttműködik az animációkezeléssel és a fegyverhasználattal.

4.2.4. Animációk és átmenetek

A karakter animációit *AnimationTree* alapú rendszer kezeli, mivel ez támogatja:

- animációk közti zökkenőmentes átmeneteket,
- több réteg (pózolás, támadás) összeadását,
- változó sebességű animációkat,
- állapotfüggő keverést, például futás közben támadást.

Az animációrendszer fontos szerepet játszik a harcban is, mivel a sebzés kiváltása általában animációs „frame” -hez van kötve (pl. sword attack 5. frame → hitbox aktiválása).

4.2.5. Státusz rendszer

A souls-like játékokban kiemelt szerepe van a *stamina*-menedzsmentnek, továbbá a játékos karakternek három fő erőforrása van:

- *Health* (HP): ha eléri a 0-t → halál → visszatöltés az utolsó ellenőrzőpontnál
- *Stamina*: támadások, futás és dodge használata *stamina*-költést igényel, a regeneráció késleltetett (mint a Souls játékokban)
- *Mana*: a mágiák használatához szükséges, lassan regenerál, viszont nem teljes mértékben.

A rendszer engedélyezi:

- maximális HP / Stamina növelését (fejlesztési menüben)
- regenerációs sebesség változtatását
- passzív vagy aktív erősítők alkalmazását

Az erőforrásokat a stat -ok növelésével lehet módosítani, a stat -ok a következők:

- *Vitality*: vitalitás, életerőt befolyásolja
- *Strength*: erő, a közelharci fegyverek sebzését módosítja.
- *Dexterity*: ügyesség, a stamina mértékét befolyásolja.
- Intelligence: intelligencia, a mágikus támadások sebzését befolyásolja.

Az erőforrások növekedéséhez egy csökkenő hozamú függvény lett alkalmazva, amely azt eredményezi, hogy egy bizonyos szinten sokkal kevesebb mértékben növeli az erőforrások maximális értékét a függvény a következő, fontos, hogy ez egy stat szintjétől függ nem pedig a karakter szintjétől:

$$\text{max\_resource} = base + (60 - base) * (\text{stat} + 20) \quad (1)$$

Szintlépésenként lehet megnövelni egy statot egyszer, de egyszerre több szintet is lehetséges lépni, ha van hozzá elég aranyunk. Minden szintlépés egyre több aranyba kerül, amit szintén egy függvénynél számítunk ki, ami a következő:

$$cost = (level + (next + abs(level - new\_level))) * 3 + 6 \quad (2)$$

4.2.6. Harcrendszer és sebzéskezelés

A harcrendszer ütközési alapú: a fegyver hitboxa aktiválódik, amikor az animáció megfelelő fázisa elérkezik, ez általában az első pár másodperc. A rendszer tartalmaz:

- aktív hitboxok, amelyek a támadás alatt aktiválódnak,
- hurtboxok, amelyek a karakter körüli sérülési zónák,

Sebzés hatására a karakter átkerül a Knockback állapotba, amely rövid animációt és kontrollvesztést okoz.

4.3. Ellenségek implementációja

Az ellenségek rendszerének kialakítása során elsődleges cél volt egy olyan rugalmas, könnyen bővíthető keretrendszer létrehozása, amely egyszerre képes kiszolgálni az egyszerűbb alapellenfeleket és az összetettebb, viselkedést igénylő főellenségeket. A rendszer a játékos karakter struktúrájához hasonlóan moduláris, így a legtöbb ellenség közös alapokra épül, viselkedésük pedig külön scriptben szabályozható.

4.3.1. Ellenségek jelenetfelépítése

A Godot jelenetalapú felépítését kihasználva minden ellenség egy közös „EnemyBase” jelenetből öröklődik. Ez biztosítja, hogy a közös komponensek minden ellenségben azonos módon működjenek. A tipikus ellenség jelenet felépítése a következő:

- CharacterBody2D / CollisionShape2D: a fizikai ütközések és mozgás alapja
- AnimatedSprite2D vagy AnimationTree: animációk kezelése
- Hitbox / Hurtbox: támadások és sérülések zónái
- RayCast2D érzékelők: játékos érzékelése, talajvizsgálat, akadályfelismerés
- Area2D érzékelők: Észlelési zóna, engedélyezi a játékos érzékelést, ha a közelben van
- State Machine script: az ellenség viselkedésének állapotokra bontása

Ez a felépítés lehetővé teszi, hogy új ellenségek minimális munkával hozzáadhatók legyenek. Jelenleg három ellenség fajta van:

- Denevér: teljes testes hitbox alapján támad

- Mágus: távolról kezd támadni, mágiákkal
- Harcos: kardal támad közelharcban

4.3.2. AI és viselkedés logika

Az ellenségek döntéshozatala egy egyszerű, de jól konfigurálható állapotgépen alapul. A legtöbb ellenfél a következő állapotokkal rendelkezik:

- Idle: alapállapot, mozgás nélkül
- Patrol: előre meghatározott útvonal bejárása
- Chase: a játékos üldözése, ha látótávolságba kerül
- Attack: közelharci támadás aktiválása
- Knockback: sebződés, animációs visszarúgás, hasonló a játékoséhoz
- Death: halál animáció, majd eltűnés

A rendszer paraméterei szerkeszthetők, így például a látótávolság, támadási rádiusz ellenségenként egyedileg állítható.

4.3.3. Ütközés és sebzés kezelés

A játékoshoz hasonlóan az ellenségek is külön kezelik, a hitbox -okat (amikor támadnak), és a hurtbox -okat. A harc logikája a következő:

1. A támadás animáció adott frame-jén a hitbox aktiválódik.
2. Ha a játékos hurtbox -át eléri, sebzés történik.
3. Sikeres találat esetén animáció, illetve esetleges knockback aktiválódik.

Ez a megoldás biztosítja a pontos, animációhoz kötött találatkezelést, amely elengedhetetlen a souls-like játékok ütemes és időzítésalapú harcrendszeréhez. Az ellenségek külön ütközési rétegen vannak ahogy a játékos is, a lényeg az volt, hogy a játkost ne blokkolják az ellenségek ezzel könnyítve a sikeres támadást.

4.3.4. Főellenségek megvalósítása

A főellenségek összetettebb viselkedést igényelnek, így külön bővített állapotgépet használnak. A fő különbségek az alapellenfelekhez képest:

- Speciális mozgásminták
- Nagyobb támadási távolság és sebzés
- Egyedi UI elemek (pl. főellenség életsáv)

A főellenség rendszere ugyanarra az Enemy osztály alapra épül, de felülírja és kiterjeszti annak funkcionalitását. Továbbá a főellenségeket csak egyszer kell legyőzni,

4.3.5. Zsákmány és interakciók

Az ellenségek legyőzés után opcionálisan dobhatnak erőforrást (a játék fejlesztési valutáját) vagy különleges tárgyakat (kulcsok, fegyverek). A zsákmány rendszer szintén moduláris, így minden ellenséghez egyedi drop -táblázat rendelhető.

4.4. Harcrendszer megvalósítása

A harcrendszer a játék egyik központi eleme, amely meghatározza az egész játékmenet ütemét és nehézségét. A souls-like műfajra jellemző, hogy minden akció – legyen az támadás, kitérés vagy védekezés – jól időzített, animációfüggő folyamatként működik, ahol a játékos és az ellenségek is szigorúan betartják a mozdulatok indulási és recovery idejét. Ennek megfelelően a harcrendszer kialakításánál elsődleges szempont volt a pontos ütközéskezelés, az átlátható támadási logika és az animációkhoz illesztett sebzéskiváltás.

4.4.1. Támadások felépítése és animációhoz kötése

A támadások nem egyszerűen gombnyomásra történő sebzéskiváltások, hanem több lépésből felépülő folyamatok. Egy tipikus támadás ciklusa:

1. Start-up fázis: a támadás előkészítése (wind-up), még nincs sebzés.
2. Active fázis: a támadás ténylegesen sebzi az ellenfelet, ekkor aktiválódik a hitbox.
3. Recovery fázis: a támadás utólagos mozdulata, amely alatt a játékos sebezhető és nem szakítható meg.

A rendszer teljes egészében animáció vezérelt, a hitbox aktiválása egy adott frame-hez van kötve, így a támadások időzítése konzisztens és jól tanulható marad. A megvalósítás során a fegyver vagy a kéz animációjára külön csomópont kerül, amely a megfelelő pillanatban jelet ad (signal), ekkor és aktiválódik a támadási hitbox.

4.4.2. Hitbox–hurtbox alapú ütközéskezelés

A harcrendszer központi eleme a hitbox–hurtbox modell:

- Hitbox: az a terület, amely támadás közben sebzést okoz.
- Hurtbox: az a terület, amely találatot kaphat.

A játékos és az ellenségek is rendelkeznek hurtboxszal, így a rendszer következetes mindkét oldal számára. A hitboxok tipikusan:

- csak a támadás active fázisában jönnek létre,
- ütközés esetén egyszeri sebzést adnak le,
- majd automatikusan kikapcsolnak.

Ez a megoldás nagyon közel áll a Souls-játékok működéséhez, ahol a találat ténylegesen a „fegyver ívének" adott pillanatához kötött, mivel pixel játékról beszélünk ezért pixel perfektek a hitboxok.

4.4.3. Stamina alapú harcmechanikák

A stamina a harcrendszer alapvető erőforrása, amely megakadályozza a végtelen folyamatos támadást. Stamina fogy támadásnál, és a dash használatakor továbbá a stamina regenerációja késleltetett, vagyis:

- csak néhány tizedmásodperccel az utolsó cselekvés után indul el,
- teljes kifogyás esetén ideiglenesen nem használhatók támadások.

Ez a rendszer arra kényszeríti a játékost, hogy megfontoltan időzítse a támadásokat.

4.4.4. Sebzés kiszámítása és visszajelzések

A sebzés meghatározásánál több tényező játszik szerepet:

- a játékos fegyverének alap sebzése
- a karakter aktuális stat -jai
- esetleges erősítők, gyengítők

A találat visszajelzései vizuális és hanghatásokat is tartalmaznak, vagyis ellenségek rövid „Knockback” animációt játszanak le, és villanó effekt vagy részecske jelzi a találatot. A pontos visszajelzés elengedhetetlen a harc „érzetéhez”, amely a souls-like műfajban kiemelten fontos.

4.4.5. Védekezés és kitérés (i-frame rendszer)

A souls-like játékok egyik jellegzetessége a kitérés (roll / dodge), amely rövid idejű sebzésmentességet biztosít. Ez az invincibility frames (i-frame) szakasz:

- animáción belül meghatározott kezdő és végző frame-ek között aktív,
- ezalatt a hurtbox ideiglenesen ki van kapcsolva,
- a kitérés stamina-költséggel jár.

A rendszer kiegyensúlyozza a harcot: a játékos nem egyszerűen „kiugrik” a helyzetből, hanem precízen időzítenie kell.

4.5. Eszköztár és tárgyrendszer implementációja

Az eszköztár (inventory) és a tárgyrendszer megvalósításakor elsődleges cél az volt, hogy egy bővíthető, adat vezérelt, könnyen karbantartható rendszer készüljön, amely jól együttműködik a játék többi alrendszerével, mint a harc, a fejlődés, a barkácsolás, valamint a HUD-on megjelenő gyorselérési kerék. Ennek megfelelően a rendszer alapját egy adatfájl-alapú tárgydefiníciós struktúra képezi, amelyben minden tárgy külön erőforrásként van tárolva.

4.5.1. Tárgyak adatvezérelt felépítése

Minden tárgyhoz külön. *tres* vagy *.res* erőforrásfájl tartozik, amely egy egységes *Item* típusból származik. A tárgydefiníció tartalmazza többek között:

- név és leírás,
- ikonkép,
- halmoz hatóság / mennyiségi limit,
- funkcionális adatmezők (pl. okozott sebzés, gyógyítás mértéke, speciális effekt, inkább a gyerek osztályokra jellemzők).
- *cooldown,* ami csak a használati tágyaknál / fegyvereknél van általában pár másodperc kivéve például *health_potion*

Ez a megközelítés lehetővé teszi, hogy a játék logikája ne függjön konkrét tárgyaktól; új tárgyak hozzáadása a rendszer módosítása nélkül megoldható. A tárgyakat egy globális *item_loader* singleton tölti be, a barkácsoláshoz szükséges receptekkel együtt. A tárgyak típusok szerint külön osztályokra vannak bontva például egy *health_potion* a *ConsumableItem* osztályba tartozik.

4.5.2. Eszköztár struktúra

Az eszköztár megvalósítása egy **listaalapú, halmozást támogató adattároló rendszer**, amely több komponensre oszlik:

- **Backend (Inventory)**: A tényleges adatkezelésért felelős réteg; ez tartja nyilván a tárgyak listáját, a mennyiségeket és a halmozható tárgyak összecsoportosítását.
- **Frontend (InventoryMenu)**: A grafikus felület, amely rácsalapú elrendezésben jeleníti meg a tárgyakat, ikonnal, mennyiséggel és kiválasztási/hozzárendelési funkciókkal.

A struktúra támogatja a jövőbeli bővítéseket, mint például fegyver effektek, tárgy ritkasági szintek vagy új UI rendszerek.

4.5.3. Gyorselérési kerék integráció

A gyorselérési kerék az eszköztár egy speciális, harc közbeni gyorselérési felülete, amely eltér a hagyományos souls-like megoldástól: a játékos **az összes rendelkezésre álló tárgyat** felteheti a kerékre, majd egyetlen dedikált gombbal (use/activate) az aktuálisan kiválasztott elemet azonnal aktiválhatja. A megközelítés célja a gyors reakciók és a taktikai variabilitás egyszerűsítése: a kerék nem csak fogyóeszközök (pl. gyógyító), hanem fegyverek és varázslatok gyors aktiválására is szolgál.

4.5.4. Tárgyfelvétel és tárgyhasználat logikája

A tárgyak felvétele és használata egységesített jelfeldolgozás alapján történik. Először megnézzük a tárgy benne van-e az eszköztárban, ha nincs hozzáadjuk, ha van akkor a már létező tárgy halmozásához hozzáadunk egyet.

A tárgyhasználat egy eseményalapú mechanizmussal kapcsolódik a játékos karakter logikájához. Például gyógyító tárgy használatakor a rendszer jelet küld a karakter Health komponensének, míg fegyver aktiválásakor a rendszer a fegyver sebzését továbbítja a harcrendszernek.

4.5.5. Barkácsolás rendszer integrációja

A barkácsolás rendszer a tárgyrendszerre épít, és annak adatait használja a receptek kezelésére. A receptek szintén erőforrásfájlokban vannak tárolva, amelyek tartalmazzák:

- a szükséges alapanyagokat,
- az elkészült tárgyakat,
- mennyiségeket,

A barkácsolási folyamat során az eszköztár adataiból kerül levonásra az alapanyag, majd a megkapott tárgy bekerül a játékos eszköztárába. Mivel a tárgydefiníciók egységesek, a barkácsolás könnyen bővíthető anélkül, hogy a rendszerhez hozzá kellene nyúlni. A szükséges, és eredmény tárgyak egy egységes erőforrás osztályban tárolódnak, ami miatt akármilyen tárgyból készíthetünk akármit, a recept maga három hibát tud jelezni, ami az *EXCEEDS_STACK_SIZE, MISSING ITEMS,* és *OK,* ez azért fontos, hogy a játékosnak jelezhető legyen UI szinten, hogy sikeres volt-e a barkácsolás

4.6. Mentés rendszer implementáció

A mentésrendszer célja egy olyan megbízható és robusztus struktúra létrehozása volt, amely összhangban áll a játék dizájnfilozófiájával: a játékos csak meghatározott pontokon – a „checkpoint” -oknál – képes menteni, és mindig a **legutolsó érvényes, nem korrupt mentés** kerül automatikusan visszatöltésre. A rendszer ennek megfelelően egyszerű, mégis biztonságos felépítést követ, különösen nagy hangsúlyt fektetve az adatvesztés és adatinkonzisztencia elkerülésére.

4.6.1. Slot-alapú mentés felépítése

Új játék indításakor a játék létrehoz egy **mentési slotot** ez a slot több mentési állapotot is tartalmazhat, de ezek **nem választhatók szabadon**; kizárólag a legfrissebb állapot tölthető be. A slot funkcionálisan a következőket tartalmazza:

- metaadatok (slot neve, létrehozás dátuma stb.…),
- 4–5 belső mentési állapot,

Egy mentési állapot a következőket tartalmazza:

- metaadatok (játékos statisztikái, mentés ideje)

- eszköztár és felszerelés állapota,
- világállapot (pl. legyőzött főellenségek-ok),
- ellenőrző pont egyedi azonosító, ami aztán a játékos pozíciója lesz,

A belső mentések szigorúan időrendben kerülnek mentésre, és a rendszer automatikusan kezeli, melyik a legfrissebb, illetve melyik sérült. Fontos, hogy nem szabad a file módosításának dátumát használni mert, ez az érték már akkor is módosul, ha megnyitódik külsőleg a fájl, a mentésben tárolni egy dátumot sokkal konzisztensebb megoldást eredményez. A fejlesztési fázis megkönnyítésének érdekében *„. tres”* fájlokat használunk *„.res”* helyett mivel az előbbi szövegesen tárolja az adatokat. Mappa struktúra szerint a *slot* -ok a mappák, és a menési állapotok a fájlok. Ez egy egyszerű átlátható rendszert képez a fejlesztés során.

4.6.2. Singleton alapú kezelő rendszer

A mentés logikai magját egy **singleton autoload** komponens valósítja meg (SaveManager), amely globális elérhetőséget biztosít minden mentéssel kapcsolatos funkció számára. A singleton a következő feladatokért felel [2, 9]:

- mentések előkészítése és érvényességellenőrzése,
- az aktuális játékállapot átadása és fogadása más rendszerektől (eszköztár, játékos adatok, térkép állapot stb.).

A mentést igénylő objektumok egy különleges *persistent* csoportba kerülnek és szükséges, definiálniuk egy „*save”* függvényt, amely egy JSON- formátumú adatot ad vissza, ami ezután alkalmas a mentésre. Ez a központosított megközelítés garantálja, hogy minden alrendszer következetes módon kommunikál a mentésrendszerrel.

4.7. Felhasználói felület megvalósítása

A felhasználói felület (UI) megvalósítása a játék egyik legfontosabb feladata volt, mivel a souls-like műfaj jelentős hangsúlyt fektet a játékos folyamatos informálására, ugyanakkor a képernyőn megjelenő elemeknek nem szabad zavarniuk a játékmenetet. A Godot Engine beépített UI-rendszere, a **Control**-alapú komponensek hierarchiája lehetővé tette egy rugalmas, adaptív és könnyen bővíthető felület kialakítását. A UI elemei több részre lettek osztva, különálló jelenetekbe és komponensekbe szervezve, hogy a moduláris felépítés megmaradjon, és minden felület külön is fejleszthető és tesztelhető legyen.

4.7.1. Főmenü

A Title Screen a játék indításakor jelenik meg, és öt fő funkciót biztosít:

- **Folytatás** (automatikusan a legfrissebb, nem sérült mentést, a legutóbb játszott slot -ból)
- **Új játék indítása**
- **Játék betöltése menü**
- **Beállítások menü**
- **Kilépés**

A felület minimalista dizájnt kapott, amely illeszkedik a műfaj komor és hangulatos megjelenéséhez. Az egyes gombok navigációja támogatja a kontrolleres és a billentyűzetes irányítást is. A betöltési menü nem jelenít meg explicit több mentést, a rendszer belső logikája határozza meg, melyik mentés tölthető be, így a játékosnak nincs szüksége manuális kezelésre.

4.7.2. Játékon belüli fő menü

A játék közben előhívható menü tartalmazza:

- a játék folytatása,
- a beállítások menü megnyitását,
- a karakter információk megtekintése
- a főmenübe való kilépést,
- a játékból való kilépést és mentést,

Mivel a játék nem támogatja a manuális mentést, a menü egyszerűsített formában működik, kizárólag a karakter státusz ellenőrzésére és a konfiguráció elérésére szolgál. A menü nem állítja meg a játékot, mint ahogy a legtöbb hasonló játékban, sőt egyik UI elem sem még az ellenőrző pontok sem [12].

4.7.3. Beállítások menü

A beállítások menü több szekcióra tagolódik a **Grafikai beállítások** (ablakmód, VSync) és a **Hangbeállítások** (Master, SFX, Zene csúszkák). A Godot *ProjectSettings* API-ját használva a

módosítások valós időben alkalmazhatók, és automatikusan elmentődnek a játék következő indításához. Néhány beállítást nem elég csak a projektbeállítás változtatásával lementeni, például az ablakmódot mivel akkor csak újraindítás után lenne érvényes [12].

4.7.4. HUD

A HUD olyan minimális, mégis informatív elemekből áll, amelyek minden souls-like játék alapvető részét képezik:

- **Életerő csík (HP bar)**
- **Állóképesség csík (Stamina bar)**
- **Varázslat csík (Mana bar)**
- **Gyűjthető erőforrás számláló** (a játék univerzális erőforrása, ez csak akkor látszik, ha módosul az értéke vagy megnyomjuk a **Z**-gombot)
- **Gyorselérési kerék megjelenítése**
- **Mini státusz effekt ikonok (ha szükséges)**

A csíkok saját egyszerű képanimáció alapján működnek, így a játékos egyértelmű visszajelzést kap a sebzés pillanatáról és mértékéről. A HUD minden eleme külön *Control node*-alapú jelenetként készült, így bármely komponens könnyen módosítható vagy lecserélhető.

4.7.5. Gyorselérési kerék megvalósítása

A gyorselérési kerék a korábban ismertetett logika alapján működik, a játékos az eszköztár bármely elemét ráteheti, majd egy forgatható kerék segítségével választhatja ki a kívánt tárgyat. A kerék a HUD részeként, de egy külön animált rétegben helyezkedik el. A UI oldalon ez a következő elemeket tartalmazza:

- körkörös elrendezésű ikonok,
- kiválasztott elem kiemelése (méretnövelés + fényhatás),
- cooldown overlay, és mennyiség számláló,

A kerék egy egyedi, a Godot beépített *Container* osztályát írja felül, aminek használatával könnyedén elkészíthető volt a körkörös elrendezése a keréknek [12].

4.7.6. Eszköztár felület

Az eszköztár külön jelenetként készült, és kategóriákra osztva jelenít meg egy listát, a UI -on lehetőség nyílik a tárgy statisztikák megjelenítésére és különböző használati műveletek kezdeményezésére, az előbb említett kategóriák a következők:

- **Közelharci fegyverek**
- **Távolharci fegyverek**
- **Fogyóeszközök**
- **Különleges tárgyak** (például pénz)

Tárgyat a kerékre rakni a tárgyra kattintással lehet, ami ezután felhoz egy nyolc részes horizontális listát amire rárakhatjuk a tárgyat, ami ezután megjelenik a keréken. A rendszer dinamikusan építi fel a listát, így új tárgytípus hozzáadása nem igényel külön UI-módosítást [12].

4.7.7. Fejlődési menü

A fejlődés menü kialakítása a Dark Souls struktúráját követi:

- jelenlegi erőforrás mennyiség,
- szintlépés költségének kijelzése,
- választható attribútumok (pl. erő, állóképesség, életerő, intelligencia stb.),
- szintlépés előtti/utáni értékek összehasonlítása.

A menü csak ellenőrző pontok használatakor érhető el, így szerves része a souls-like ritmusnak [12].

4.7.8. Barkácsolás menü

A barkácsolás felület lehetővé teszi:

- receptek listázását,
- hiányzó vagy meglévő hozzávalók megjelenítését,
- anyagszükséglet automatikus levonását sikeres barkácsolás esetén.

A UI szigorúan a backend által szolgáltatott adatokból dolgozik, így az összes tárgy, recept és követelmény egyetlen központi adatforrásból származik, a játékos eszköztárából.

4.7.9. Ellenőrzőpont menü

Az ellenőrzőpont menü aktiváláskor még nem aktiválja a pihenést, de regisztrálja a már látogatott pontokhoz, hogy gyorsutazást lehessen indítani a pontra, a menü négy elemet tartalmaz:

- **Az ellenőrzőpont egyedi neve,**
- **Pihenés** (Rest, lényebében egy betöltő képernyő jelenik meg, csak *„Resting”* felirattal),
- **Barkácsolás menü megnyitása** (Blacksmith),
- **Fejlesztési menü** (Level up),

Az ellenőrzőpont UI indítja a „Pihenés” gombbal az ellenségek újra helyezését, így a menürendszer és a játéklogika szorosan összekapcsolódik [12].

4.7.10. Gyorsutazás menü

A gyorsutazás menü az ellenőrzőpont egyik almenüje, ami egy külön lapon jelenik meg, mint ahogy a barkácsolás menü és a karakter fejlődési menüje, ez egy egyszerű lista a már látogatott pontokról. Az egyes pontok megnyomásával azonnal a kívánt ponthoz tudunk gyors utazni, vagyis, módosítjuk a karakter pozícióját a kívánt pont helyének közelébe [12].

4.8. Pálya és világ megvalósítása

A játék pályájának és világának megtervezése és implementálása kulcsfontosságú a játékos élmény szempontjából. A világ nem csupán vizuális környezetet biztosít, hanem a játékmenet alapját is meghatározza: útvonalakat, akadályokat, ellenfelek elhelyezését, interakciós pontokat, valamint azokat a területeket, ahol a játékos különböző rendszerekkel — például harccal, felfedezéssel — találkozik. A világ megvalósítása Godot -ban több komponensből áll: térbeli felépítés, navigációs rendszer, ütközéskezelés, világobjektumok, világlogika.

4.8.1. Világstruktúra

A világ kialakítását a Godot *Tilemap* csomópontja segítségével épül fel, és segíti a struktúrák, objektumok lehelyezését. Az objektumok és más komplex elemek jelenetenként vannak lehelyezve, erre beépített opció van, ami nagyban megkönnyítette a pálya elrendezését.

4.8.2. Navigációs rendszer (Navigation + NavigationMesh)

A navigációs rendszer és az előbb említett *Tilemap* használatával, van elkészítve egy üres cella, ami arra szolgál, hogy az ellenfelek melyik területeken navigálhatnak. Ezek a cellák külön rétegen vannak annak érdekében, hogy ne egységesen egy helyen legyen, és hogy ne ütközzenek a többi réteg fizikai ütközési testeivel. A platformok, sima cellaként vannak megvalósítva, csak az ütközési testük van módosítva, hogy *one-way* legyen, így garantálva a helyes működését.

4.8.3. Interaktív és destruktív elemek

A játék világába helyezett interaktív objektumok (összetörhető tárgyak, csapdák, ládák) célja kettős, egyrészt játékmeneti kihívást és taktikai döntési helyzeteket teremtenek, másrészt változatosságot és felfedezési motivációt adnak a nyitott pályának. A ládák egyszerűen véletlen generált tárgyakat adnak egy globális scriptből, ami nyilván tartja a tárgyakat, a törhető elemek csak esztétikai célt szolgálnak semmi több, úgy, mint a láncok, amik a plafonról lóghatnak, viszont a létra már funkcionális célt szolgál, hogy komplexebb pálya dizájnokat lehessen tervezni a jövőben.

5. TESZTELÉS

A tesztelés célja a játéklogika megbízhatóságának, helyességének és stabilitásának biztosítása. Mivel a projekt egyik kritikus eleme a karakterfejlesztési mechanika (attribútumok növelése, költségkalkuláció, erőforrás-felhasználás). Ezen felül alkalmaztunk integrációs és kézi playtesteket is, amelyek a rendszerek teljes együttműködését vizsgálják.

5.1. Tesztelési stratégia

- **Unit tesztek**: elsősorban a karakterfejlesztési rendszer numerikus és logikai számításaira koncentrálnak (pl. szintlépés költségének számítása, stat -átalakítások, erőforráslevonás). Ezek automatizált, izolált tesztek, amelyek nem futtatják a teljes játékot, csak a kalkulációs logikát ellenőrzik.

- **Integrációs tesztek**: a fejlődési menü, adatbetöltés és mentésrendszer közti együttműködés tesztelése (például, ha fejleszt a játékos, helyesen frissül-e a stat és a mentés).

- **Manual / playtesting**: balansz- és élménytesztek, ahol valós játékosok próbálják ki a mechanikát, hogy észrevegyük az olyan problémákat, amelyeket automata tesztek nem fognak fel (pl. megérzéses nehézség, UI érthetőség).
- **Edge-case tesztek**: nem csak normál folyamatokat tesztelünk, hanem szélsőséges bemeneteket (pl. 0 erőforrás, nagyon nagy értékek, párhuzamos hívások), hogy a rendszer robusztus maradjon.

5.2. Tesztelendő modulok (prioritások)

1. Költségszámítás (level up / skill cost): a legtöbb regresszió itt okoz problémát → magas prioritás.
2. Stat-alkalmazás: új értékek helyes kiszámítása és alkalmazása a karakterre (hp_max, stamina_max).
3. Erőforrás-kezelés: resource counter levonása, visszaállás hibák kezelése.
4. Limitálások: max/min értékek, negatív bemenetek kezelése.
5. Interfész a mentéssel: a fejlesztés állapota mentésbe kerül-e és betöltődik-e helyesen.

5.3. Tesztkörnyezet és eszközök

- Godot + GDScript teszteket futtathatunk, egyszerű, saját teszt futtató scripttel, amely GDScript-függvényeket hív és assertions -t hajt végre.
- Tesztadatok tárolása: izolált, fix inputok (mock adatok) és fixture-k alkalmazása (pl. előre definiált karakterállapotok).

ÖSSZEFOGLALÁS

A Szakdolgozat megvalósításával egy olyan játékot sikerült elkészítenem, ami egyesíti a platformer játékok mozgás rendszerét a souls-like játékok nehéz mechanikáival. A projekt során kiemelt szempontot kapott a modularitás, a bővíthetőség és az áttekinthető architektúra kialakítása, amely lehetővé teszi a játék egyszerű továbbfejlesztését és karbantartását.

Bízom benne, hogy a bemutatott megoldások — különösen az állapotgép-rendszer, a modularitásra építő architektúra és a Godot motor sajátosságaira építő implementációk — más projektek számára is hasznos kiindulópontot vagy inspirációt jelenthetnek. A játék jelenlegi formájában szilárd alapot biztosít a további tartalmak, mechanikák és funkciók fejlesztéséhez, és jól szemlélteti, hogyan épül fel egy korszerű, rugalmas 2D akció-RPG rendszer.

A projekt során az AI használata, többségében a karakter stat-ok, illetve szintlépéshez használt függvények elkészítéséhez, illetve nagyrészben rendszerek működésének optimalizálására, vagy szimplán ötletelési szempontból volt használva, generált kód csak minimális esetben fordul elő a projektben, és ezek a kódok is legtöbbször átírásra/ellenőrzésre kerültek.

A játék jelenlegi állapota stabil alapként szolgál további tartalmak – például komplexebb harcrendszer, új ellenféltípusok, részletesebb világok, valamint történet és küldetésrendszer – hozzáadásához. A szakdolgozat bemutatta, hogyan lehet a Godot-motorban egy professzionális, jól strukturált és hatékony 2D akció-RPG rendszert felépíteni, amely egyszerre fejlesztőbarát és játékosbarát megoldásokat alkalmaz. A projekt teljes forrása megtekinthető az alábbi github repository -ban: https://github.com/Warlord56x/SZTE-TTIK-Szakdolgozat.

IRODALOMJEGYZÉK

[1] https://godotengine.org/ utolsó megtekintés: 2025. 12. 09.

[2] https://docs.godotengine.org/en/stable/ utolsó megtekintés: 2025. 12. 09.

[3] https://docs.godotengine.org/en/stable/classes/index.html#nodes utolsó megtekintés: 2025. 12. 09.

[4] https://docs.godotengine.org/en/stable/classes/index.html#resources utolsó megtekintés: 2025. 12. 09.

[5] https://orama-interactive.itch.io/pixelorama utolsó megtekintés: 2025. 12. 09.

[6] https://en.wikipedia.org/wiki/Godot_(game_engine) utolsó megtekintés: 2025. 12. 09.

[7] https://en.wikipedia.org/wiki/Soulslike utolsó megtekintés: 2025. 12. 09.

[8] https://gameprogrammingpatterns.com/ utolsó megtekintés: 2025. 12. 09.

[9] https://gameprogrammingpatterns.com/singleton.html utolsó megtekintés: 2025. 12. 09.

[10] https://gameprogrammingpatterns.com/state.html utolsó megtekintés: 2025. 12. 09.

[11] https://freesound.org/ utolsó megtekintés: 2025. 12. 09.

[12] https://gameuidatabase.com/ utolsó megtekintés: 2025. 12. 09.

[13] https://docs.godotengine.org/en/stable/tutorials/2d/2d_sprite_animation.html utolsó megtekintés: 2025. 12. 09.

[14] https://docs.godotengine.org/en/stable/tutorials/export/exporting_projects.html utolsó megtekintés: 2025. 12. 09.

[15] https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html utolsó megtekintés: 2025. 12. 09.

[16] https://en.wikipedia.org/wiki/Platformer utolsó megtekintés: 2025. 12. 09.

[17] https://docs.godotengine.org/en/stable/tutorials/2d/custom_drawing_in_2d.html utolsó megtekintés: 2025. 12. 09.

KÖSZÖNETNYÍLVÁNÍTÁS

Ezúton szeretném megköszönni mindenkinek, aki hozzájárult a szakdolgozatom elkészítésében, külön kiemelném témavezetőmet, Dr. Jász Judit egyetemi adjunktust a szakmai tanácsaiért, útmutatásáért. Továbbá hálás vagyok minden a Godot játékmotort fejlesztő programozónak, családomnak és nem utolsó sorban barátaimnak.

NYILATKOZAT

Alulírott, Török Dániel, programtervező informatikus BSc szakos hallgató, kijelentem, hogy a szakdolgozatban ismertetettek saját munkám eredményei, és minden felhasznált, nem saját munkából származó eredmény esetén hivatkozással jelöltem annak forrását.

Szeged, 2025. december 09. _____

Aláírás