Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Papp Márton

2022

Szegedi Tudományegyetem
Informatikai Intézet

2D–s platformer játékfejlesztés

Szakdolgozat

Készítette:
Papp Márton
gazdaságinformatika szakos hallgató

Témavezető:
Dr. Nagy Antal
egyetemi docens

2022

Feladatkiírás

Egy 2-dimenziós platformer játék fejlesztése Unity-ben. A játékban megvalósításra került a játék menetének irányítása, pályák betöltése, a karakterek kidolgozása, valamint az aktuális eredmény kijelzése. A projekt részét képezi, egy a játékon belül megvalósított pályaszerkesztő, amely képes az adott pályák betöltésére, módosítására, mentésére, valamint újak létrehozására is.

Tartalmi összefoglaló

- A téma megnevezése:

2D–s platformer játékfejlesztés

- A megadott feladat megfogalmazása:

A feladat egy 2-dimenziós platformer játék elkészítése Unity-ben, melyben a játékos különböző pályákat teljesíthet, amelyeket a beépített pályaszerkesztőben módosíthat, törölhet, vagy újakkal bővíthet.

- Alkalmazott eszközök, módszerek:

AUnity által biztosított 2-dimenziós játékfejlesztéshez rendelkezésre álló eszközök, valamint a hozzájuk tartozó logikai háttér megvalósítása C# nyelven. A kész játék telepítőjéhez az InnoSetup nevű, szkriptalapú telepítőkészítő szoftver.

- Elért eredmények:

A kész játék bármilyen Windows 10, vagy újabb verziójú Windowst-t futtató hardveren játszható, amennyiben az rendelkezik az irányításához szükséges perifériákkal (egér, billentyűzet).

- Kulcsszavak:

2D, platformer, játék, Unity, C#

Tartalom

1. Bevezetés

A platformer játékok, olyan játékok, melyekben a játékos egy főhőst irányít, akinek egy általában egyszerű fő küldetése van. Ahhoz azonban, hogy ezt elérje különböző akadályokon kell átverekednie magát, ezzel színesebbé és élvezhetővé téve az amúgy egyszerű elemekből álló játékot [1].

Szakdolgozatomban olyan játék elkészítése volt a cél, amivel bárki szívesen játszik és nem igényel hosszabb megszokást, vagy beletanulást. Az egyszerű 2dimenziós játékokok nagy többsége ebbe a kategóriába esik, hiszen nagy részük a videójátékok őskorában készült, amikor még ilyen egyszerűbb játékok is komoly feladványt jelentettek a fejlesztőiknek a rendelkezésre álló hardverek és szoftverek korlátjainak köszönhetően. A projekt készítésekor igyekeztem ennek a kornak a szellemét megjeleníteni a játékomban, és egy nosztalgikus hangulatú játékot létrehozni.

A fő célom egy egyszerű könnyen kezelhető, jól átlátható játék fejlesztése volt. Amelyben a felhasználó a beépített pályákat a saját ízlésére szabhatja, mind hosszúságban, mind azok nehézségében. Teheti mindezt a játékban megvalósításra került pályaszerkesztőben, amely, lehetőséget biztosít a pályán lévő összes elem módosítására, valamint újak hozzádására. Az elkészült pályákat pedig bármikor újra lehet játszani, vagy tovább módosítani.

2. Játék elvárások

A játék elkészítésekor a kiírásnak megfelelő szoftvert kellett létrehoznom. A megfelelő működésen kívül több, különböző elvárás volt megfogalmazva.

Az egyik fő elvárás a menü és a karakter teljeskörű irányításának lehetősége volt. A játékba indítása teljes képernyős módban következik be. A felhasználó a kurzor irányításával valamint jobb- és bal egérkattintással navigálhat a menürendszerben, amely a következő elemekből áll. Főmenü, játéktér, pályaszerkesztő, beállítások menü, Pause menü, valamint a pályák betöltéséhez és mentéséhez használt menü. A pályák mentésekor, a pályanevet a billentyűzet segítségével adhatjuk meg. Ezen kívül a fő karakter irányításának lehetősége is

alapvető volt. A karakter irányítása szintén a billentyűzet segítségével történik. A <W>,<A>,<S>,<D> valamint a <Ctrl> és <Space> gombok hatására.

Az irányítás mellett fontos hangsúly esett a pályák felépítésére és azok testreszabhatóságára. Ennek köszönhetően a játéknak rendelkezni kellett egy a szoftveren belül megvalósított pályaszerkesztővel is, amelyben a felhasználó a meglévő pályákat módosíthatja, vagy teljesen új sajátokat is létrehozhat. A szerkesztőben az összes pálya minden részletét módosíthatja a felhasználó, valamint a mozgó részeket valós időben követheti.

A pályaszerkesztővel szorosan egybefüggő logika a pályák mentésének és betöltésének lehetősége. Ennek szintén a játékon belül szimplán a pályanév megadására vagy az adott pálya nevének kiválasztására kell történnie. A pályákat az játékot futtató gépen lokálisan tároljuk, úgy, hogy azok a játékból kilépve is megmaradnak, és legközelebb is hozzáférhetőek lesznek.

A játékmenethez kapcsolódó elvárások közé tartozik a különböző hangok, hangeffektek megléte is. Ezek nagyban befolyásolják a játék élvezhetőségét, és meghatározzák a játékvilág atmoszféráját. Fontosnak tartottam, hogy a különböző eseményekhez köthető hangokon kívül a játék rendelkezzen egy háttérzenével is, hogy teljesebb képet kapjon a játékos.

3. A játék megvalósítása

3.1 Fejlesztési és tesztelési környezet

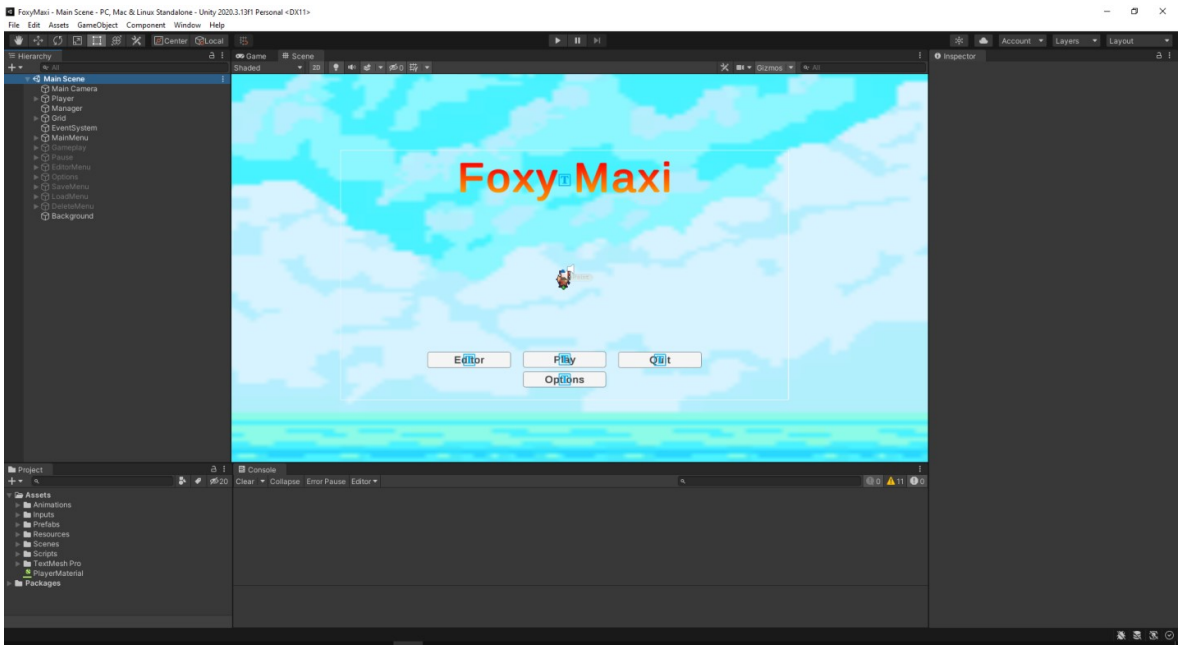
A szoftver elkészítéséhez a Unity 2020.3.13f1 verzióját használtam. A Unity egy játékfejlesztő platform, melynek segítségével 2D és 3D multiplatform élményeket és játékok készítésére ad lehetőséget [2]. A Unity egy MacOS játékmotorként indult még 2005-ben, később folyamatosan egyre több platformon vált elérhetővé, köztük PC, mobil, konzol és virtuális valóság eszközökön is. Különösen népszerű az Android és iOS játékfejlesztők körében a felhasználóbarát és könnyű kezelhetősége miatt [3].

A játék logikájához tartozó scripteket C# nyelvben írtam meg. Ezeket a fájlokat a *Visual Studio 2019 Community Edition IDE*-ben szerkesztettem. A *Visual Studio* egy számítógépes szoftverek, weboldalak, web alkalmazások, web szolgáltatások és mobil alkalmazások

fejlesztésére használt fejlesztői környezet, amely magába foglalja a kódszerkesztőt és a hozzá tartozó debuggert mind gépi és forráskód szinten is. A szoftver támogatja az F#, C++, C#, Visual Basic programozási nyelveket, valamint az XML leíró nyelvet is. A Visual Studio általam használt Community Edition kiadása ingyenes tanulóknak, nyílt-forráskódú fejlesztőknek és magánembereknek [4,5].

3.2 Unity bevezetés

Az új projekt létrehozása után az Unity fejlesztői környezetében találjuk magunkat, amely több ablakból és komponensből áll össze. Az ablak felső részében találhatjuk a szalagmenü-t, alatta az irányításhoz szükséges módok gombjait. Az ablak nagy részét több, tetszés szerint mozgatható kisebb ablak adja. Helyet kap a játékbeli objektumok listája és az ezek paramétereit mutató ablak, a projekthez tartozó fájlokat mutató fájlmenedzser, egy konzol ablak, valamint egy nézet ablak, ahol magát a játékot és annak különböző nézeteit láthatjuk.



3.2 ábra: Unity kinézete

A Unity-ben készített játékok több komponensből épülnek fel. Ezek közül a leglapvetőbb a játékelem azaz *GameObject*. A játékelemeket rendezhetjük különböző színterekbe. Az általam készített játékban az összes objektum egy fő szintéren belül található. Ahhoz, hogy a játék indításakor a képernyőn megjelenjen valami egy Kamera típusú *GameObject*-re van szükség. Ezt, mint minden más játékelemet egy 3 dimenziós térben helyezzük el, annak

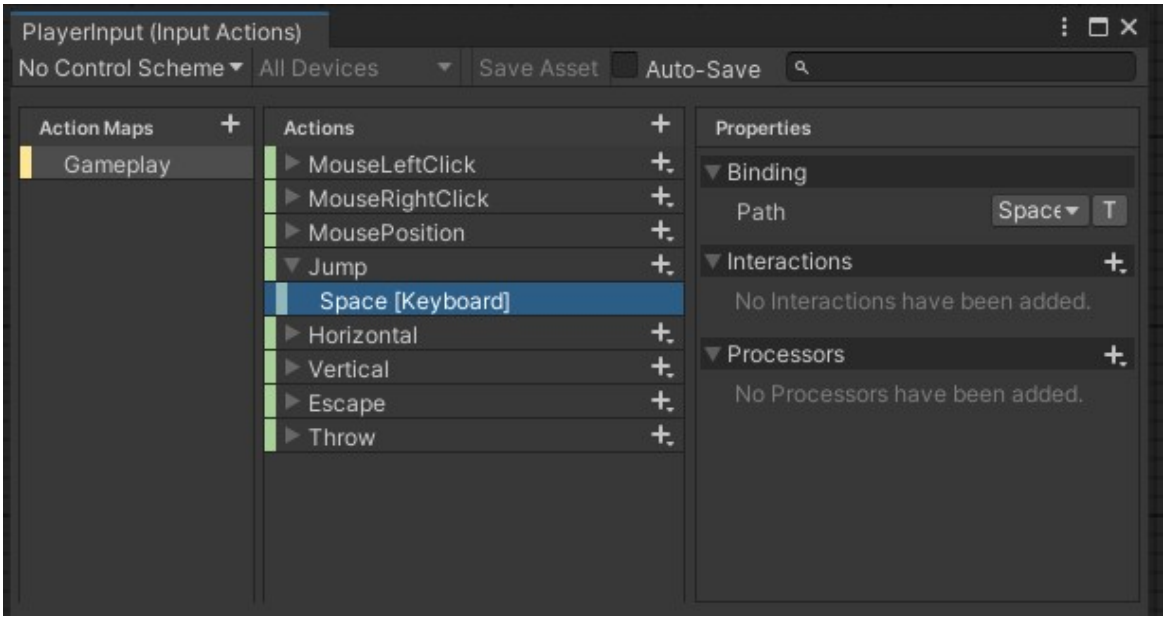
ellenére, hogy a játék csak 2 dimenziós. A játékelemekhez különböző komponenseket rendelhetünk. Jelen esetben a kamerához van rendelve még egy *AudioListener* komponens is, ami a többi *GameObject*-hez tartozó komponensek által keltett hangot dolgozza fel és továbbítja a felhasználó felé [6,7,8,9].

3.3 Főhős

A játékban a főhős is egy *GameObject* amelyet a kamera előtt, annak látóterében helyezkedik el. Az un. *Player* objektum rendelkezik egy *SpriteRenderer* nevű komponenssel, amely az objektum megjelenítését szolgálja. Itt került beállításra a főhős kinézete, amely png kiterjesztésű képsorozat egy képe [10].

3.3.1 Irányítás

A főhős irányításához először a Unity-ben le kell kezelni a felhasználó által adott jeleket, melyeket a két fő perifériával, az egérrel és a billentyűzettel adhatunk meg. Ahhoz, hogy ezekhez a jelekhez akciókat lehessen rendelni a Unity *New Input System*-jét használtam. Ezt először a beépített *Package Manager*-ből telepítettem. A telepítés után létre kellett hozni egy *Input Action Asset*-et, amely egy beállítások fájlt generált, melyben a később beállított akciók automatikusan frissülnek. Ezután az *Action Editor*-ban beállíthatók lettek a különböző akciók és a hozzájuk tartozó beviteli jelek.



3.3.1 ábra: Action Editor

Ezután a Player objektumhoz egy új script file komponenst adtam, ami a karakter irányításáért volt felelős. A C# scriptben összekötöttem a különböző gombokhoz tartozó akciókat a Player objektummal. Az objektum rendelkezik egy *Rigidbody2D* típusú komponenssel, amellyel elérhető, hogy hassanak rá a fizika törvényei, így például a gravitáció [11]. A főhős mozgatása az adott objektumhoz adott erővel történik, az alábbi módon:

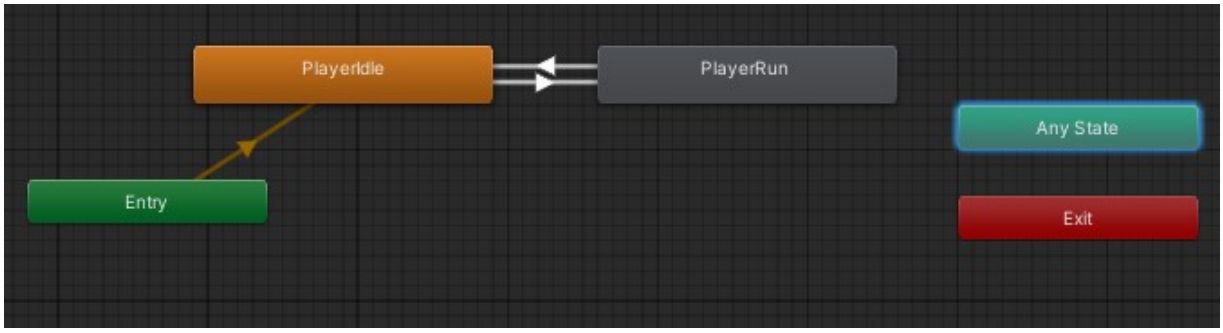
```
PlayerInput imp;
private Rigidbody2D rb;

void Start()
    {
        imp = new PlayerInput ();
        rb = this.GetComponent<Rigidbody2D>();
    }

if (imp.Gameplay.Jump.triggered)
    {
        Jump();
    }

private void Jump()
    {
        rb.AddForce(new Vector2(0, jumpForce), ForceMode2D.Impulse);
    }
```

A karakter jobb- és bal irányba való mozgatása hasonló elven működik, azonban amennyiben a játékos irányt változtat, a képernyőn való megjelentése tükrözésre kerül, hogy mindig arccal a menetirány felé legyen fordulva.

```
private void flip()
    {
        facingRight = !facingRight;
        Vector3 theScale = this.transform.localScale;
        theScale.x *= -1;
        this.transform.localScale = theScale;
    }
```

3.3.2 Animáció

A főhős kétfajta animációval rendelkezik. Az egyik álló helyzetben, a másik pedig futás közben látható. Az animációkat egy a főhős objektumához adott *Animator* komponens segítségével hoztam létre. A komponensnek szüksége van egy *Animator Controller*-re, amelyben az animációk sorrendjét és azok egymáshoz való viszonyát kell megadni.



3.3.2 ábra: Animator Controller

Az *Animator Controller*-ben lévő animációkat egy beépített szerkesztővel hozhatjuk létre. Egy idővonalon tetszőleges időpontokhoz különböző megjelenítendő képet választhatunk, ezzel animációt kreálva. Az elkészült animációkat az Animator-on keresztül hívhatjuk meg az objektumhoz tartozó szkriptben.

```
private Animator anim;

void Start()
{
    anim = this.GetComponent<Animator>();
}

//Player animation
anim.SetFloat("speed", Mathf.Abs(this.rb.velocity.x));
```

3.3.3 Dobás

A főhős a pályákon található ellenségeket egy tárgy dobásával képes megsemmisíteni. A tárgyak dobását egy a *Player* objektumhoz rendelt scripttel oldottam meg. Ahhoz, hogy a színtérben lévő objektumok egymásra erőket tudjanak kifejteni és össze tudjanak ütközni szükségük van egy *Rigidbody2D* nevű komponensre. A scriptben egy ilyet hoztam létre. Amennyiben a változó elérhetőségét *public* (bárhonnan elérhető)-re állítjuk, az objektumhoz tartozó *Inscpector*-ban bármilyen típus azonos objektumot hozzárendelhetünk. Ebben az esetben egy *Prefab* típusú objektumot rendeltem hozzá. A *Prefab*-ok játékobjektumok tárolására alkalmas fájlok, amelyekben az objektum összes komponensével kerül tárolásra [12]. Ezen kívül a kódban létrehoztam egy *Rigidbody2D*-ből álló listát, amiben a *Prefab*-okat tárolom. Amikor a script betöltődik, lefut, a Start függvény ahol a lista feltöltődik a dobálandó tárgyakkal. Amikor a főhős eldobja a tárgyat, az a listából kikerül, majd az eldobás után egy bizonyos idő elteltével újra a listába kerül, a színtérből pedig eltűnik. Ahhoz, hogy az eldobott tárgy dobáskor ne akadjon bele a főhősbe, az főhős objektumához,

de annak területén kívül egy másik játékobjektumot hoztam létre, amely követi a főhős pozícióját. A tárgyak innen kerülnek eldobásra.

```
public Rigidbody2D itemProto = null;
public float speed = 1.0f;
public Transform firePoint;
public int itemPoolSize = 10;
public List<Rigidbody2D> itemPool;

void Start()
{
    itemPool = new List<Rigidbody2D>();
    for (int i = 0; i < itemPoolSize; i++)
    {
        Rigidbody2D itemClone = Instantiate(itemProto);
        itemClone.gameObject.SetActive(false);
        itemPool.Add(itemClone);
    }
}

void Update()
{
    if (imp.Gameplay.Throw.triggered)
    {
        throwItem();
    }
}
void throwItem()
{
    Rigidbody2D itemClone = getItemFromPool();
    itemClone.transform.position = firePoint.position;
    itemClone.gameObject.SetActive(true);
    float direction = playerController.facingRight ? +1 : -1;
    Vector3 force = transform.right * speed * direction;
    itemClone.velocity = force;
}

private Rigidbody2D getItemFromPool()
{
    foreach (Rigidbody2D item in itemPool)
    {
        if (!item.gameObject.activeSelf)
        {
            return item;
        }
    }
    return null;
}
```

3.3.4 Életerő

A főhős rendelkezik egy adott mennyiségű életerővel, amelyet minden pálya betöltésénél a maximum értékre állítunk. Az életerő két esemény hatására csökkenhet, amennyiben a játékos leesik a pályáról, és amennyiben hozzáér egy ellenséghez. A pályáról való leeséskor a főhős pozícióját figyeli egy script, amely ha egy bizonyos szint alá esik, akkor a főhős életereje a maximum életerő harmadával csökken. Az ellenségekkel való ütközés során vesztett életerő pedig a főhőshöz és az ellenfelekhez is hozzáadott *Collider2D* komponens segítségével valósul meg. A *Collider 2D* megadja, hogy az adott objektum beállított kiterjedésén belül hozzáér e másik objektum *Collider*-éhez. Amennyiben az érintett ütköző egy ellenséghez tartozik, a főhős életerejét csökkentjük [13].

```
private Collider2D objectCollider;
public static int Health = 200;

void Start()
{
    objectCollider = this.GetComponent<Collider2D>();
}

void Update()
{
    if (objectCollider.IsTouchingLayers(LayerMask.GetMask("Enemy")))
    {
        Health--;
    }
}
```

3.3.5 Életerő kijelzése

A játék közben a főhős életerejét valós időben követhetjük egy, a bal felső sarokban elhelyezett jelzősávon. A sáv két egymásra helyezett játékobjektumból áll. Az alsó egy szimpla fehérrel kitöltött téglalap, a felső pedig egy piros hátterű 1 pixelből álló kép. A felső objektumhoz adott *Image* komponens tartalmazza a képet, amelynek a *FillAmount* változóját állítva a piros kép balról jobbra takarja el az alatta lévő fehér hátteret. A felső objektumhoz tartozó script *Update* metódusában változtatjuk a kép kitöltésének változóját. Az *Update* minden képernyőre kirajzolt új képkockánál lefut, ezáltal valós időben változtatja az előbb említett változót egy a főhőshöz tartozó másik scriptben lévő szám alapján, amelyben a főhős aktuális életereje van tárolva.

```
private Image HealthBar;
public static float CurrentHealth;
private float MaxHealth;

private void Start()
{
    MaxHealth = PlayerHealth.Health;
    HealthBar = GetComponent<Image>();
}

private void Update()
{
    CurrentHealth = PlayerHealth.Health;
    HealthBar.fillAmount = CurrentHealth / MaxHealth;
}
```

3.4 Kamera

A játék futtatása közben a képernyőn megjelenő részeket a kamera objektumon keresztül látjuk. A játékban egyetlen kamera található, melynek kimenete az elsődleges monitor.

3.4.1 Kamera mozgása

Amennyiben a főhős játékobjektuma aktív, a kamera azt követi, így mindig a főhős van a képernyő középpontjában. Ez egy a kamera objektumához tartozó script segítségével valósul meg. A főhős pozícióját a hozzá tartozó objektum *Transform*-jából elérhető. Minden játékobjektum rendelkezik *Transform*-mal, ebben tárolódik az objektum pozíciója, forgatása és skálázása. Ezen kívül egy változóban tárolódik az előző pozíció (*anchorPos*), és az új pozíció előzőhöz számított különbsége (*offset*). A script ezek alapján állítja a kamerát mindig az új pozícióra [14]. A kamera mozgatásának sebességét a *speed* változó segítségével lehet változtatni.

```
public float speed = 1.0f;
public Transform target;
public Vector3 offset;

void Start()
{
    offset = target.position - transform.position;
}
```

```
void Update()
{
    if(target) {
        Vector3 anchorPos = transform.position + offset;
        Vector3 movement = target.position - anchorPos;

        Vector3 newCamPos = transform.position + movement * speed * Time.deltaTime;
        transform.position = newCamPos;
    }
}
```

A pályaszerkesztőben a kamera pozícióját a felhasználó állíthatja a <W>,<A>,<S>,<D> billentyűkkel.

```
if (InEditor)
        {
            float xAxisValue = imp.Gameplay.Horizontal.ReadValue<float>() * 0.1f;
            float yAxisValue = imp.Gameplay.Vertical.ReadValue<float>() * 0.1f;

            transform.position = new Vector3(transform.position.x + xAxisValue, transform.position.y +
            yAxisValue, transform.position.z + 0);
        }
```

3.5 Ellenségek

A pályák teljesítése közben két fajta ellenséggel találkozhatunk. Az álló ellenség mindig ugyanabban a pozícióban marad és sebezi a főhőst, ha az hozzáér. A másik típusú ellenfél a mozgó ellenfél, a nevéből adódóan folyamatosan mozog.

3.5.1 Ellenségek mozgása

A mozgó ellenfél a rendelkezésére álló egybefüggő sík területen végez folyamatos oda-vissza mozgást. Amennyiben előtte elfogynak a platformok, vagy falba ütközik, megfordul és az ellenkező irányba halad tovább. Ezeket az eseteket három *Raycast* metódus segítségével detektálja. A *Raycast* a Unity Physics könyvtárában található segédmetódus. Egy boolean visszatérési értékkel rendelkezik. Egy a paraméterében megadott hosszúságú sugarat bocsájt ki, egy paraméterben megadott pontból, egy szintén paraméterben megadott irányba. Amennyiben a sugár egy másik objektum *Collider*-ével ütközik, pozitív értéket ad [15]. A három sugár a főhős lába előtt lefelé, valamint a közepétől két oldalirányba tart. Ezek alapján fordul meg abban az esetben amennyiben a jelenlegi irányba nem tud tovább haladni.

```
public class EnemyPatrol : MonoBehaviour
{
    public float speed;
    public float rayDistance;
    public Transform GroundWallDetection;
    private bool movingRight = true;
    private Rigidbody2D rb;

    void Start()
    {
        rb = this.GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        RaycastHit2D groundCheck = Physics2D.Raycast(GroundWallDetection.position,
        Vector2.down, rayDistance);
        RaycastHit2D wallCheckRight = Physics2D.Raycast(GroundWallDetection.position,
        Vector2.right, rayDistance/10);
        RaycastHit2D wallCheckLeft = Physics2D.Raycast(GroundWallDetection.position,
        Vector2.left, rayDistance/10);

        if (movingRight && groundCheck)
        {
            rb.velocity = Vector2.right * speed;
        }
        else if (!movingRight && groundCheck)
        {
            rb.velocity = Vector2.left * speed;
        }

        if (rb.velocity.y==0)
        {
            if (movingRight && (groundCheck.collider == false || wallCheckRight))
            {
                flip();
                movingRight = false;
            }
            else if (!movingRight && (groundCheck.collider == false || wallCheckLeft))
            {
                flip();
                movingRight = true;
            }
        }

    private void flip()
    {
        movingRight = !movingRight;
        Vector3 theScale = this.transform.localScale;
        theScale.x *= -1;
        this.transform.localScale = theScale;
    }
}
```

3.5.2 Ellenségek sebzése

Az ellenfelek képesek sebezni a főhőst, amikor hozzáérnek. Ez a főhős és az ellenfelek *Collider*-ének használatával valósul meg egy a főhőshöz tartozó scriptben. Amennyiben a főhős *Collider*-je egy másik *Enemy* rétegbe tartozó objektum *Collider*-évél ér össze, a főhős életereje csökkenni kezd.

```
void Update()
    {
        if (objectCollider.IsTouchingLayers(LayerMask.GetMask("Enemy")))
        {
            Health--;
            Debug.Log("Oh no, I hurt myself! HP: " + Health);
        }
    }
```

3.6 Menürendszer

A felhasználó a játékban különböző menüpontokat és képernyőket érhet el, amelyek között az egér segítségével tud navigálni.

3.6.1 Menürendszer általános működése

Minden menü, valamint a pályaszerkesztő és a játéktér is rendelkezik egy saját *GameObject*-tel. Ezekhez az objektumokhoz tartozik egy *Canvas* nevű objektum, melynek a feladata az, hogy a felhasználói interfész (*UI*) elemeit tartalmazza. Ide taroznak a különböző gombok, legördülő menük, szövegek, stb [16]. A *Canvas Scaler* segítségével a benne található elemek több különböző felbontású képernyőre is arányosan skálázhatóak. A játékban a különböző menük mind egy *Scene*-ben találhatóak, ha nincs rájuk szükség inaktív állapotba kerülnek.

```
public class MainMenu : MonoBehaviour
{
    public GameObject GameplayUI;
    public GameObject MainMenuUI;
    public GameObject LoadMenuUI;

    public void PlayButtonPressed()
    {
        MainMenuUI.SetActive(false);
        LoadMenuUI.SetActive(true);
        GameplayUI.SetActive(true);
        LoadMenu.LoadFromEditor = false;
    }
}
```

3.6.2 GameOver képernyő

A pályák teljesítése közben, ha a főhős életereje elfogy,a képernyőn megjelenik a „Game Over” felirat, majd három másodperc múlva a *Pause* menü. A script inaktívvá teszi az éppen aktív elemeket, és csak a feliratot teszi aktív státuszba. Az *Invoke* függvény segítségével egy másik függvény kerül meghívásra, a függvény paramétere a hívandó függvény, valamint egy float típusú szám, amely a hívás idejét adja meg (3 másodperc). Az idő elteltével, az így meghívott *SetFalse* függvény hatására kerül a felhasználó a *Pause* menübe.

```
public class PlayerHealth : MonoBehaviour
{
    public GameObject pauseButton;
    public GameObject GameOverText;
    public GameObject PauseButton;
    public GameObject HealthBar;
    public GameObject player;

    public static int Health = 200;
    private int OriginalHealth = 200;

    void Update()
    {
        if (Health <= 0)
        {
            PauseMenu.GameOver = true;
            Health = OriginalHealth;
            Prefabs.DestroyPrefabs();
            player.gameObject.SetActive(false);
            GameOverText.gameObject.SetActive(true);
            PauseButton.gameObject.SetActive(false);
            HealthBar.gameObject.SetActive(false);
            Invoke("SetFalse", 3.0f); // disable after 3 seconds
        }
    }
    void SetFalse()
    {
        GameOverText.gameObject.SetActive(false);
        pauseButton.GetComponent<Button>().onClick.Invoke();
        player.gameObject.SetActive(true);
        PauseButton.gameObject.SetActive(true);
        HealthBar.gameObject.SetActive(true);
    }
}
```

3.7 Pályaszerkesztő

A játékban egy beépített pályaszerkesztő használható, amely új pályák létrehozását, valamint a meglévők módosítását is lehetővé teszi a játékosok számára. A szerkesztő felépítése az alábbi módon működik.

3.7.1 Tilemap

A Unity-ben készített 2D játékokhoz tartozó pályák felépítésének alapja az úgynevezett *TileAsset*. Ezek egyenlő méretű négyzet alakú objektumok (csempék), melyekhez tartozik egy *Sprite*, ami lényegében egy képfájl. A *Tilemap* ezeket a részeket rendezi egy rácsos szerkezetbe [17]. A játékban több *Tilemap* is található. Az egyik a pálya elrendezését tartalmazza. Ide töltődik be minden pálya. Ahhoz, hogy a főhős a *Tilemap*-ban található platformokon közlekedhessen a főhős és a *Tilmeap* is rendelkezik egy *Collider2D* komponenssel.

A pályaszerkesztőben a felhasználó minden csempe helyére tehet egy platformelemet, ezekkel kialakítva a pályákat. A pályaszerkesztőben egy gombbal választható ki a letenni kívánt platformelem. A kiválasztás után a felhasználó egy előnézetet láthat a kurzor helyén a letenni kívánt objektum másával. Ez az előnézet egy másik, azonos méretű *Tilemap*-ban lehet látni az objektum lehelyezéséig [18,19].

```
private void Update()
    {
        if (selectedObj != null) //if something is selected - show preview
        {
            Vector3 pos = _camera.ScreenToWorldPoint(mousePos);
            Vector3Int gridPos = previewMap.WorldToCell(pos);

            if (gridPos != currentGridPosition)
            {
                lastGridPosition = currentGridPosition;
                currentGridPosition = gridPos;

                UpdatePreview();
            }
        }
    }

    private void UpdatePreview()
    {
        previewMap.SetTile(lastGridPosition, null); // Remove old tile if existing
        previewMap.SetTile(currentGridPosition, tileBase); // Set current tile to current mouse positions tile
    }
```

Amennyiben a kiválasztott platformelemet bal kattintással a kiválasztott mezőbe helyezi a felhasználó, a Tilemap-ben az adott pozícióhoz tartozó objektum frissítésre kerül.

```
private void OnLeftClick(InputAction.CallbackContext ctx)
    {
        if (selectedObj != null && !EventSystem.current.IsPointerOverGameObject())
        {
            DrawItem();
        }
    }

private void DrawItem()
{
    // only draw items within specified map size
    if (floorMap.GetTile(currentGridPosition) != null)
    {
        defaultMap.SetTile(currentGridPosition, tileBase);
    }
}
```

3.7.2 Prefab

Hasonlóan a főhős által eldobható tárgyhoz, a pályákon található ellenségek, valamint a pálya kezdetét- és végét jelző zászlók is *Prefab*-okból jönnek létre. A felhasználó szintén egy gombbal választhat az objektumok közül, majd a kiválasztott objektum bal kattintásra az kurzor helyén jön létre. A kiválasztás után itt is látható egy előnézett az választott objektumról, amely az *ItemImage* változóban kerül tárolásra [20].

```
public class PrefabEditorManager : MonoBehaviour
{
    public PrefabController[] PrefabButtons;
    public GameObject[] ItemPrefabs;
    public GameObject[] ItemImage;
    public int CurrentButtonPressed;
    PlayerInput imp;
    Vector2 mousePos;

    void Update()
    {
        Vector2 screenPos = new Vector3(mousePos.x, mousePos.y);
        Vector2 worldPos = Camera.main.ScreenToWorldPoint(screenPos);
        // placing prefab
        if (imp.Gameplay.MouseLeftClick.triggered && PrefabButtons[CurrentButtonPressed].Clicked)
        {
            PrefabButtons[CurrentButtonPressed].Clicked = false;
            Instantiate(ItemPrefabs[CurrentButtonPressed], new Vector3(worldPos.x, worldPos.y, 0),
Quaternion.identity);
            Destroy(GameObject.FindGameObjectWithTag("ItemImage"));
        }
    }
}
```

A Prefab alapú pályaelemek mozgásra is képesek, valamint, mivel szinte teljes értékű játékobjektumok, *Rigidbody* és *Collider* komponensekkel is rendelkeznek. A törlésükhöz a

létrehozott objektumokat kell eltávolítani. Ez a *RaycastHit2D* nevű beépített függvény segítségével valósul meg. A *RaycastHit2D*első paramétere a felhasználó kurzora, a második pedig az általa kilőtt sugár iránya. A függvény alapján ezáltal lekérdezhetőek a kurzor helyén lévő objektumok, valamint azok *Collider*-ei. Amennyiben ezek egy Start vagy Cél zászlóhoz tartoznak, azok számát a törléskor eggyel növelni kell, hiszen ezekből egy pályán csak egy fordulhat elő.

```
if (imp.Gameplay.MouseLeftClick.triggered && deletePrefab)
{
    RaycastHit2D hit = Physics2D.Raycast(Camera.main.ScreenToWorldPoint(mousePos), Vector2.zero);

    if (hit.collider != null)
    {
        if (hit.collider.gameObject.tag == "Start")
        {
            start                                                                    =
GameObject.FindGameObjectWithTag("StartFlagButton").GetComponent<PrefabController>();
            start.IncreaseQuantity();
        }
        if (hit.collider.gameObject.tag == "Finish")
        {
            end = GameObject.FindGameObjectWithTag("EndFlagButton").GetComponent<PrefabController>();
            end.IncreaseQuantity();
        }
        Destroy(hit.collider.gameObject);
        deletePrefab = false;
    }
}
```

3.7.3 Mentés

A pályák mentése két részből áll. A program játék külön menti a *Tilemap* és a *Prefab*-ok adatait egy-egy fájlba. A *Json* file (**J**ava**S**cript **O**bject **N**otation) egy kulcs-érték párokat tartalmazó szöveges dokumentum, amelyet adatok tárolására és szállítására használnak [21].

A Tilemap mentésekor először a csempék bejárásával egy láncolt lista jön létre, amely tartalmazza az összes nem üres pozíción lévő csempék adatait.

```
public void GetWorldTiles (Tilemap tileMap, string SaveName, bool save)
    {
        saveTiles = new List<WorldTile>();

        foreach (Vector3Int pos in tileMap.cellBounds.allPositionsWithin)
        {
            var lPos = new Vector3Int(pos.x, pos.y, pos.z);

            if (!tileMap.HasTile(lPos)) continue;
```

```
            WorldTile _tile = new WorldTile()
            {
                localPlace = lPos,
                gridLocation = tileMap.CellToWorld(lPos),
                tileBase = tileMap.GetTile(lPos).name,
            };

            if(save)
            {
                saveTiles.Add(_tile);
            }
            else
            {
                tiles.Add(_tile.gridLocation, _tile);
            }
        }

        if(save)
        {
            TileMapDataSystem.Save(SaveName, "Map", saveTiles);
        }
    }
```

Ezután a lista átadásra kerül a *TileMapDataSystem* osztály *Save* függvényéhez, amely egy külső *JsonHelper* segítségével egy fájlba menti az adatokat [22]. A mentett fájl nevét a felhasználó adja meg egy szövegdobozban.

```
public static void Save (string saveName, string folderName, List<WorldTile> saveTiles)
    {
        string path = Path.Combine(Application.persistentDataPath, "Saves");

        if(!Directory.Exists(path)){
            Directory.CreateDirectory (path);
        }
        path = Path.Combine(path, folderName);
        if(!Directory.Exists(path)){
            Directory.CreateDirectory (path);
        }

        path = Path.Combine(path, saveName + ".humble");
        string saveJson = JsonHelper.ToJson(saveTiles.ToArray(), true);

        File.WriteAllText(path, saveJson);
    }
```

A *Prefab*-ok mentése ugyanezen az elven működik, azzal a különbséggel, hogy amikor a lista létrehozásra kerül, az összes játékobjektumból kerülnek kiválasztásra a megfelelő objektumok.

```
public static List<MyPrefab> GetPrefabs()
    {
        List<MyPrefab> myPrefabs = new List<MyPrefab>();
        GameObject[] allObjects = UnityEngine.Object.FindObjectsOfType<GameObject>();
        foreach (GameObject obj in allObjects)
        {
            if (obj.tag == "Start" || obj.tag == "Finish" || obj.layer == 7)
            {
                MyPrefab prefab = new MyPrefab()
                {
                    Type = obj.gameObject.tag,
                    Position = obj.transform.position,
                };
                myPrefabs.Add(prefab);
            }
        }
        return myPrefabs;
    }
```

3.7.4 Betöltés

A pályák betöltése a két mentett fájlból történik a mentett *Json* fájlok listává alakításával, majd a *Tilemap* és a *Prefab* objektumok hozzáadásával. A felhasználó egy lenyíló listából választhatja ki a betölteni kívánt pályát. A lenyíló menü feltöltése az alábbi módon történik:

```
public Dropdown LoadFiles;

public static string[] getMapFiles()
{
    string path = Path.Combine(Application.persistentDataPath, "Saves/Map/");
    string[] filePaths = Directory.GetFiles(@path, "*.humble");
    return filePaths;
}

void OnEnable()
{
    var mapFiles = getMapFiles();
    LoadFiles.options.Clear();
    foreach (string c in mapFiles)
    {
        LoadFiles.options.Add(new Dropdown.OptionData() { text =
System.IO.Path.GetFileNameWithoutExtension(c) });
    }
}
```

A csempék betöltése:

```
public void LoadWorldTiles(string TileMapName, string FolderName)
    {
        obstacleMap.ClearAllTiles();
        saveTiles = TileMapDataSystem.Load(TileMapName, FolderName);
        SetWorldTiles(obstacleMap, "Obstacle");
    }
```

```
public void SetWorldTiles (Tilemap tileMap, string folderName){
        tiles = new Dictionary<Vector3, WorldTile>();
        string path = Path.Combine("Tilemap", folderName);
        Tile[] tileAsset = Resources.LoadAll<Tile>(path);

        foreach(WorldTile tile in saveTiles)
        {
            for(int i = 0; i <= tileAsset.Length; i++)
            {
                if(tileAsset[i].name == tile.tileBase)
                {
                    tileMap.SetTile(tile.localPlace, tileAsset[i]);
                    i = tileAsset.Length;
                }
            }

            WorldTile _tile = new WorldTile()
            {
                localPlace = tile.localPlace,
                gridLocation = tile.gridLocation,
            };
            tiles.Add(_tile.gridLocation, _tile);
        }
        Resources.UnloadUnusedAssets();
    }
}
```

Az ellenségek betöltése:

```
public static void LoadPrefabs(string saveName, bool inGame)
    {
        GameObject[] ItemPrefabs = PrefabEditorManager.PublicItemPrefabs;
        List<MyPrefab> myPrefabs = PrefabDataSystem.Load(saveName, "Prefab");

        if (ItemPrefabs != null && myPrefabs != null)
        {
            foreach (MyPrefab mp in myPrefabs)
            {
                foreach (GameObject ip in ItemPrefabs)
                {
                    if (mp.Type == ip.tag)
                    {
                        Instantiate(ip, new Vector3(mp.Position.x, mp.Position.y, 0), Quaternion.identity);
                    }
                }
            }
        }
    }
```

3.7.5 Törlés

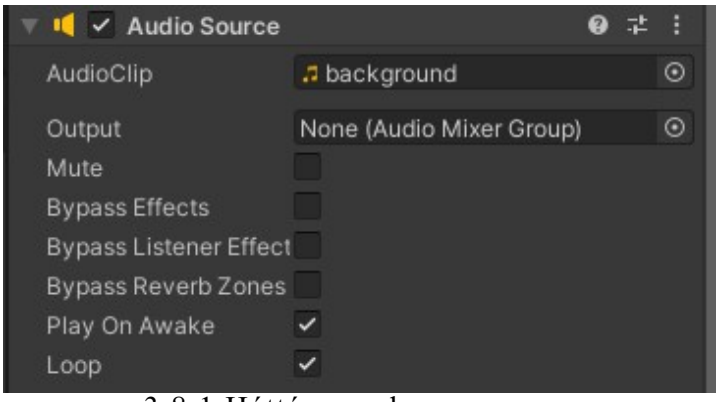
A pályákat tartalmazó fájlok törlésére is van lehetőség a pályaszerkesztőn belül. A kívánt pályát egy lenyíló menüből választhatja ki a felhasználó. A pályához tartozó fájlok a Delete gomb megnyomásával azonnal törlésre kerülnek.

```
public static void Delete(string folderName, string saveName)
    {
        string path = Path.Combine(Application.persistentDataPath, "Saves");
        path = Path.Combine(path, folderName);
        path = Path.Combine(path, saveName + ".humble");

        if (File.Exists(path))
        {
            File.Delete(path);
        }
    }
```

3.8 Hangok

3.8.1 Háttérzene

A játékban megtalálható háttérzene már a játék indítását követően megszólal és végtelenítve szól, addig, míg ki nem lépünk abból. A zene egy mp3 file, ami a *Manager* nevű játékobjektum komponense [23]. A Manager mindig aktív a játék színterében, így a zene folyamatosan szól.



3.8.1 Háttérzene komponense

A folyamatos lejátszást a *Loop* aktiválása teszi lehetővé, amellyel a zene végtelenítve játszódik le. A *Play On Awake* funkció pedig, a komponenshez tartozó játékobjektum aktiválódásakor kezdi el a lejátszást, ami jelen esetben a játék indítása.

A háttérzene a pályaszerkesztőbe való belépéskor leáll, majd onnan kilépve újraindul. A belépéskor meghívott függvény scriptjében található *AudioSource* referencia alapján kerül mindez szabályozásra.

```
public AudioSource bgmusic;
public void EditorButtonPressed()
{
    bgmusic.Stop();
}
```

3.8.2 Események hangjai

A pályák teljesítése közben a játékos négy különböző hanggal találkozhat, melyek a játékos ugrásánál, a tárgyak dobásánál, valamint a játék vége képernyő megjelenésénél hallhatóak.

3.8.2.1 Ugrás hangja

Amennyiben a főhős irányítható állapotban van a játék futásakor, a felhasználó a <Space> gombbal bírhatja őt ugrásra. Az audiófájl forrása a Player objektum komponense, amely a *PlayerController* nevű script *Update* metódusában kerül hívásra.

```
public AudioSource jump_sound;

void Update()
    {
        if (imp.Gameplay.Jump.triggered)
        {
            Jump();
            jump_sound.Play();
        }
    }
```

3.8.2.2 Dobás hangja

Az ugráshoz hasonlóan a főhős dobásához is tartozik hang, amely az hasonló módszerrel kerül lejátszásra, a különbség csupán annyi, hogy az *AudioSource*komponens nem közvetlen a játékos objektumához, hanem a hozzá tartozó *FirePoint* nevű objektumhoz tartozik.

```
public AudioSource throwsound;

void Update()
    {
        if (imp.Gameplay.Throw.triggered)
        {
            throwItem();
            throwsound.Play();
        }
    }
```

3.8.2.3 Játék vége hang

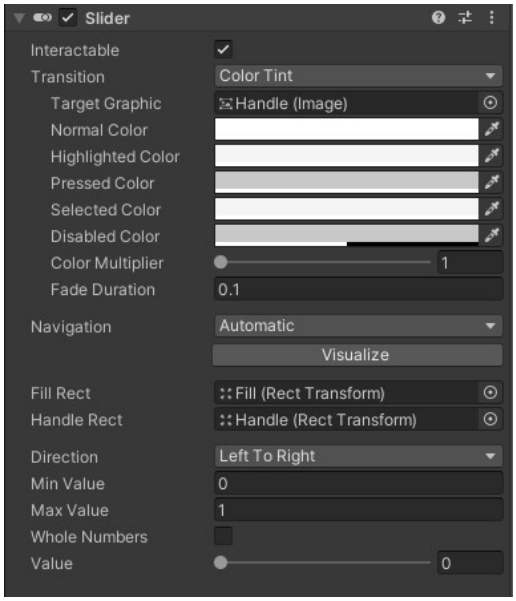
Amennyiben a pályák teljesítése közben a főhős életereje elfogy, a Játék vége képernyő megjelenésével egyidejűleg a játékhoz tartozó háttérzene leállításra kerül és a játék vége hang kerül helyette lejátszásra.

```
public AudioSource bgmusic;
public AudioSource gameover_sound;

void Update()
    {
        if (Health <= 0)
        {
            bgmusic.Stop();
            gameover_sound.Play();
        }
    }
```

3.8.3 Hangerő szabályzása

A felhasználó a játék futása közben változtathatja annak hangerejét. A változtatás az összes eseményhez tartozó hangra és a háttérzenére egyaránt vonatkozik. A főmenüben található *Options* gombra kattintva megjelenő csúszkán lehet beállítani a kívánt értéket. A csúszka balra húzásával a játék némításra kerül.

A csúszka egy *Slider* komponens segítségével valósul meg melynek a minimum és maximum értéke szabadon módosítható, jelen esetben 0,1 értékekre állított.



3.8.3 ábra: Hangerőszabályzó komponense

A hangerőt az *OptionsMenu* komponenshez tartozó script állítja be a csúszka értéke alapján. A script update metódusa folyamatosan figyeli a csúszkához tartozó értéket, amelyet kezdetben 1-re állítunk, és mindig arra az értékre állítja az *AudioListener*hangerejét, ami a csúszka értéke, ezáltal változtatva a hangerőt.

```
public class OptionsMenu : MonoBehaviour
{
    public Slider VolumeSlider;

    private void Start()
    {
        VolumeSlider.value = 1.0f;
    }

    public void AdjustVolume(float newVolume)
    {
        AudioListener.volume = newVolume;
    }

    void Update()
    {
        AdjustVolume(VolumeSlider.value);
    }
}
```

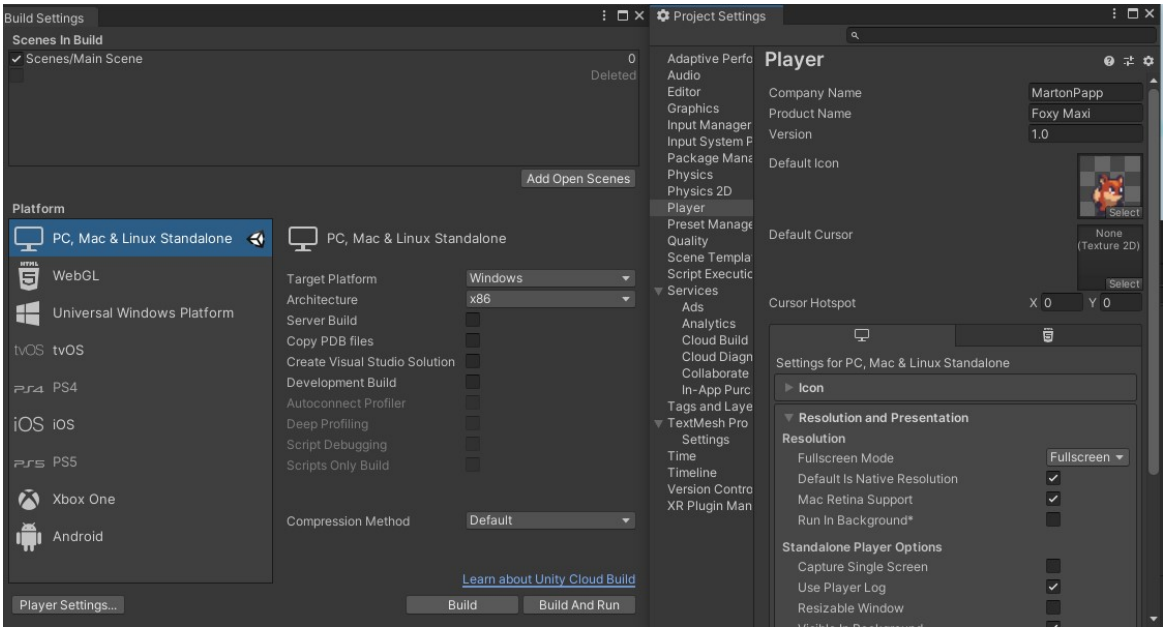
4. Játék telepítője

A játékhoz tartozik egy teljesen grafikus felületről vezérelhető telepítő, amellyel bármely Windows 10, vagy újabb verziójú Windows operációs rendszerre telepíthető pár kattintással. A telepítő elkészítése az Inno Setup Compiler 6.2.0 verziójú szoftverrel valósult meg. Az InnoSetupCompiler egy ingyenes, nyílt forráskódú telepítőkészítő amelyet Jordan Russell és MartijnLaankészített [24].

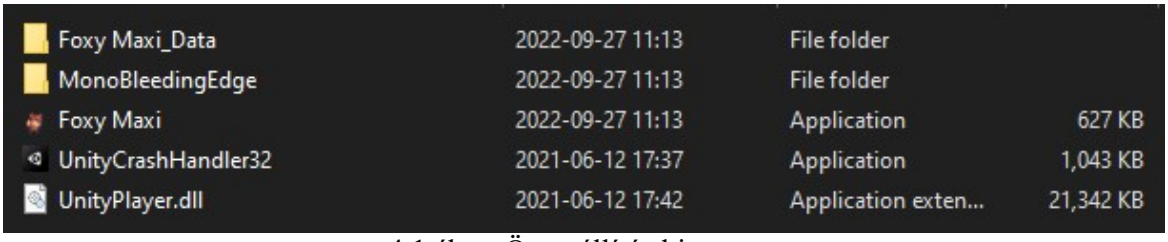
4.1 Játék összeállítása

A telepítő elkészítéshez először a Unityben található projekt összeállítására (*Build*) van szükség. Az összeállítás beállításai az alapértelmezett formában maradtak, a projekthez tartozó beállításokban pedig a Build és projektbeállítások ábrán látható adatok kerültek beírásra.

Az összeállítás után az alábbi Összeállítás kimenete ábrán látható fájlok és mappák jönnek létre. A játék már ilyenkor önállóan a Unity-n kívül is futtatható állapotban van, azonban a pályák tárolásának módja miatt, azok csak ugyanazon gépen érhetőek el. Ezt a problémát hivatott áthidalni a telepítő, amely nem csak a játékot, hanem a hozzá tartozó pályákat is felmásolja a felhasználó gépére, ezzel kiváltva azt, hogy ezt manuálisan kelljen megtenni.



4.1 ábra: Build és projektbeállítások



4.1 ábra: Összeállítás kimenete

4.2 Telepítő készítése

A telepítő készítése az Inno Setup varázslójának segítségével könnyen eszközölhető. Az első lépésben a telepítőhöz tartozó adatokat (név, kiadó, verziószám) adjuk meg, majd a telepítés helyét, ezután az előbb összeállított programot és a hozzá tartozó fájlok helyét adjuk meg, majd a kész telepítőnek egy kimeneti mappát definiálva meg is kapjuk a telpítőt. Az így elkészült telepítő azonban még mindig nem másolja fel a gépre a pályákat tartalmazó mappákat. Az elkészült telepítőhöz tartozik egy script amely a varázslóban beállított paramétereket tárolja.

4.3 Telepítő script

A végleges telepítő script a varázsló által generált változat kiegészítettje. A pályák mentésekor a Unity *Application.persistentDataPath* helyére kerültek mentésre a pályákhoz tartozó fájlok. Ennek a helye Windows alkalmazások esetében a

%userprofile%\AppData\LocalLow\<companyname>\<productname> elérési úton található [25].

A telepítő scriptben ennek az elérési útnak a megtalálását egy külső Pascal script végzi. A Pascal script a telepítőhöz tartozó script elején került beágyazásra a *Code* mezőben [25].

A telepítő script *Files* része felelős a számítógépre másolandó és telepítendő fájlokért. Az itt beszúrt új sorban került definiálásra a pályákat tartalmazó mappa a *Source* részben, valamint a külső scriptben található *GetAppDataLocalLow* függvény által visszaadott elérési út a *DestDir* részben. Ezáltal a telepítő az adott fájlokat a fent említett mappába másolja futtatáskor.

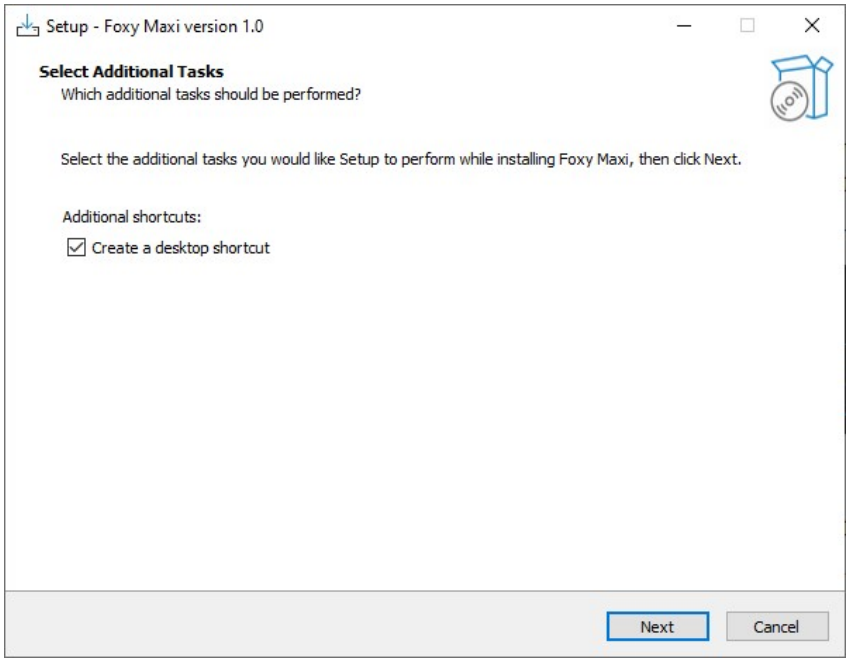
```
[Files]
Source: "C:\Users\User\Documents\Unity Projects\Builds\foxymaxi\{#MyAppExeName}"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\User\Documents\Unity Projects\Builds\foxymaxi\UnityPlayer.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\User\Documents\Unity Projects\Builds\foxymaxi\data\*"; DestDir: "{app}"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\User\Documents\Unity Projects\Builds\foxymaxi\localow\*"; DestDir: "{code:GetAppDataLocalLow}"; Flags: ignoreversion recursesubdirs createallsubdirs
```

Ezután a Unity által összeállított fájlok a scriptben megjelölt helyekre kerültek. A *FoxyMaxi_Data* és a *MonoBleedinEdge* mappák a *data* mappába, a pályákat tartalmazó mappák a *localow* mappába, a többi fájl pedig a scripttel megegyező helyre. Az így lefuttatott script kimenete pedig az előre beállított mappában megjelenő telepítő fájl, amely már megfelelően telepíti a játékot a hozzá tartozó pályákkal együtt.

5. Felhasználói leírás

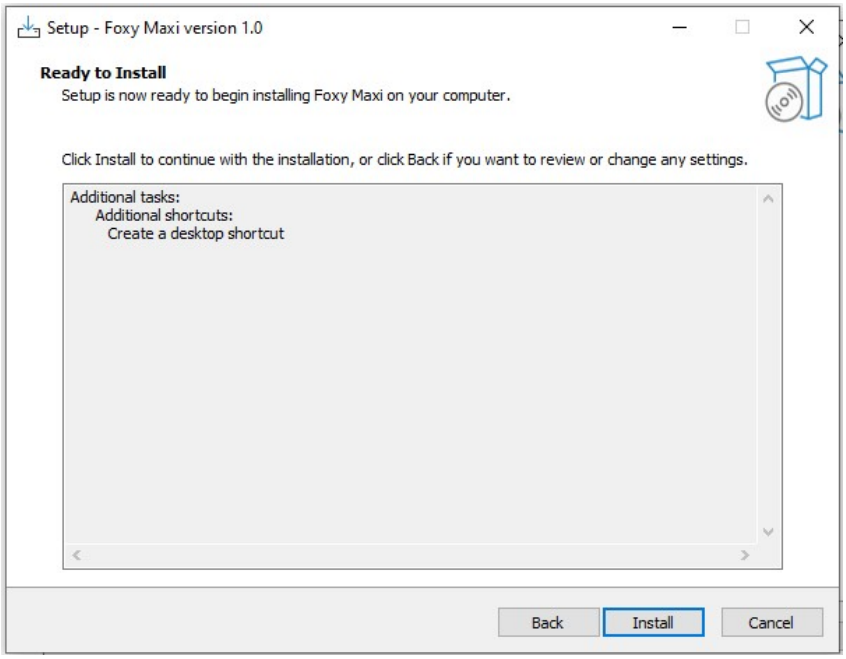
5.1 A játék telepítése

A játék telepítése a mellékelt telepítő futtatásával valósul meg. A telepítő megnyitása után a felhasználó kiválaszthatja, hogy kíván e a játéknak egy parancsikont létrehozni az asztalon. A kívánt opció kiválasztása után a *Next* gombra kattintva egy újabb képernyő jelenik meg.



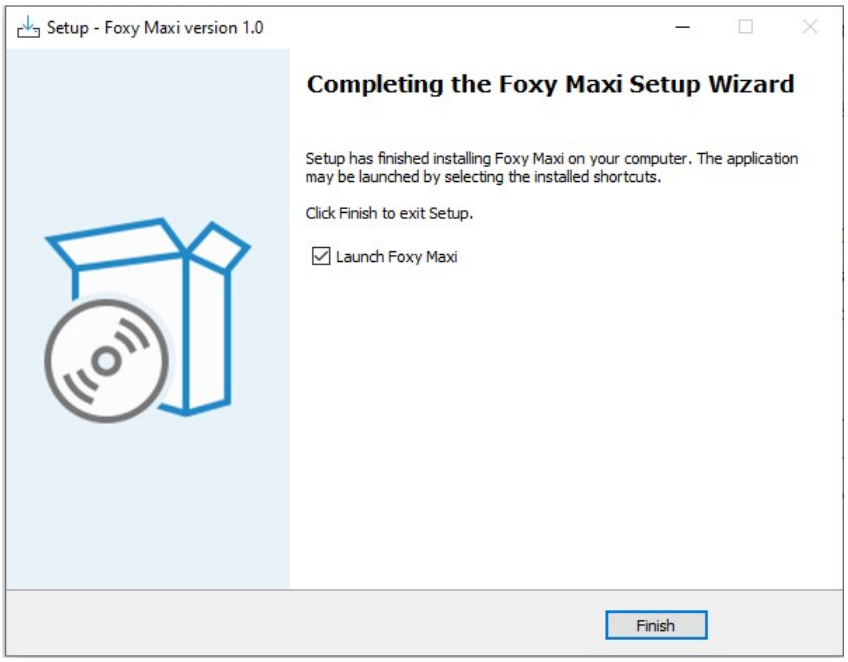
5.1 ábra: Telepítő első képernyője

A második képernyő listázza, hogy a felhasználó kívánt e parancsikont létrehozni, valamint tájékoztatja a felhasználót, hogy a telepítés megkezdéséhez minden készen áll. A *Install* gombra kattintva a telepítő lefut.



5.1 ábra: Telepítő második képernyője

A sikeres futást követően a felhasználó választhat, hogy a telepítő elhagyásával egyidejűleg elindítja e a frissen telepített játékot. A telepítő a *Finish* gombra kattintva bezárul.



5.1 ábra: Telepítő utolsó képernyője

5.2 A játék felépítése

A játékba belépve a fő menüben találjuk magunkat, ahol a képernyő közepén látható a fő karakter. Felette a játék címe olvasható, az alsó sorban pedig négy gomb közül választhatunk. Az elsővel a pályaszerkesztőbe léphetünk, a másodikkal a játékot indíthatjuk el, a harmadikkal kiléphetünk vissza az operációs rendszerbe, a negyedikkel pedig a beállításokat érhetjük el. A felhasználó a gombokat a bal egérbillentyű lenyomásával aktiválhatja, valamint a billentyűzet segítségével, itt is irányíthatja a főhőst, aki mindig a képernyőn, közepén fog elhelyezkedni. Az indítással egy időben elindul a háttérzene is, amely a pályaszerkesztő kivételével minden menüben hallható.

Az *Options* gombra kattintva a hangerő beállításokat érhetjük el. A játék hangerejét egy csúszka segítségével módosíthatjuk. A menüből az *OK* gomb megnyomásával térhetünk vissza a fő menübe. A beállított hangerő szintjét a játék kilépés után nem tárolja.

5.2 ábra: Ábra a játékról

Amennyiben a *Play* gombra kattintunk, egy menüben találhatjuk magunkat ahol az összes rendelkezésünkre álló pálya közül egyet kiválaszthatunk egy lenyíló menüből az egér segítségével. Miután a kívánt pálya kiválasztásra került, a *Load* gombra kattintva léphetünk a pályára. A betöltött pályán a főhősünk a pálya elejét jelző piros zászlóhoz kerül. A képernyő bal felső sarkában egy piros sávban láthatjuk a főhős életerejének szintjét.

A főhős irányítási ilyenkor a <W>, <A>, <S>, <D>, billentyűkkel történhet. Ugrani A <Space> (szóköz), billentyűvel, támadni pedig a <Ctrl> lenyomásával lehet. Amennyiben a játék közben a főhős hozzáér bármely ellenséghez, az életereje csökkenni fog. Szintén csökken az életerő, ha a karakter a „leesik” a pályáról, tehát nincs már alatta több platform, amin lábat vethetne. A főhős különböző akcióihoz hangok vannak rendelve, így ugráskor és dobáskor is különböző hangokat hallhatunk.

5.2 ábra: Ábra az irányításról

A pályák játszása közben a jobb felső sarokban található *Pause* gombra kattintva, vagy a billentyűzet *Escape* gombját lenyomva az úgynevezett *PauseMenu* nyílik meg, melyben a Játékot tudjuk folytatni (*Resume*), egy másik pálya betöltését kezdeményezhetjük (*Load*), vagy a fő menübe léphetünk vissza.

A pályák teljesítésének feltétele, hogy a főhős elérje a pályán tetszőleges pontban elhelyezett fekete-fehér célzászlót. Amennyiben ez megtörténik a megnyíló menüben a felhasználó újabb pálya betöltésére, valamint a fő menübe való visszatérésre kap lehetőséget.

Amennyiben a főhős életerje minimum szint alá csökken, megjelenik a képernyőn egy „Game Over” felirat, valamint a hozzá tartozó hangot hallhatjuk. A felirat 3 másodperc után eltűnik, majd a pálya teljesítésekor megjelenő menüben találhatjuk újra magunkat, ahol a már előbb ismertetett lehetőségek közül választhatunk.

A pályák teljesítése közben két fajta ellenséggel találkozhatunk. Az álló ellenség mindig ugyanabban a pozícióban marad és sebezi a főhőst, ha az hozzáér. A másik típusú ellenfél a mozgó ellenfél, a nevéből adódóan folyamatosan mozog. A rendelkezésére álló sík terepen folytonos oda-vissza mozgást végez. A mennyiben falnak ütközne, vagy nincs tovább járható

út abba az irányba, megfordul és az ellenkező irányba halad tovább. Nem képes ugrani sem lefele tartó mozgást végezni.

Amennyiben a fő menüben az „Editor” gombra kattintottunk a pályaszerkesztőbe lépünk, ahol a háttérzene megáll. A pályaszerkesztőben a karakter irányításhoz használatos billentyűkkel mozgathatjuk az építeni kívánt pályát. A bal felső sarokban található „Back” gombbal a fő menübe térhetünk vissza. A képernyő alsó felében a rendelkezésünkre álló építőelemekből készíthetjük el a pályákat. Az elemeket egy bal kattintással választhatjuk ki, majd a pályán szintén bal kattintással az adott helyre tehetjük. A pályán lévő elemek törlésére külön gombok találhatóak az építőelemekkel megegyező sorban. A képernyő jobb felső sarkában a pálya mentésre, betöltésére és törlésére használatos gombok helyezkednek el. A *Save* gombra kattintva a pályaszerkesztő jelenlegi állását menthetjük el a felugró menüben a pályához tartozó tetszőleges név megadásával, majd a *Save* gomb megnyomásával. Amennyiben egy már meglévő pályán szeretnénk módosítani, azt először a *Load* gombra kattintva, majd a pálya nevét kiválasztva tudjuk megtenni. Ha egy meglévő pályát szeretnénk törölni, a *Delete* gombra kattintva választhatjuk ki a megnyíló menüben a törölni kívánt pályát, majd a *Delete* gombbal törölhetünk. Az éppen a szerkesztőben lévő pályát a szerkesztőből a *Clear* gombra kattintva törölhetjük. Fontos, hogy ilyenkor csak az éppen aktuális változtatások vesznek el a pályák maguk nem törlődnek a felhasználó számítógépéről.

6. Irodalomjegyzék

Platformjáték (Wikipédia)

[1] https://hu.wikipedia.org/wiki/Platformj%C3%A1t%C3%A9k

Unity hivatalos weboldal

[2] https://unity.com/download

Unity (game engine) Wikipedia

[3] https://en.wikipedia.org/wiki/Unity_(game_engine)

Visual Studio hivatalos weboldal

[4] https://visualstudio.microsoft.com/free-developer-offers/

Visual Studio Wikipedia

[5] https://en.wikipedia.org/wiki/Visual_Studio#Community

Unity hivatalos dokumentáció

[6] https://docs.unity3d.com/Manual/GameObjects.html

[7] https://docs.unity3d.com/Manual/CreatingScenes.html

[8] https://docs.unity3d.com/Manual/Cameras.html

[9] https://docs.unity3d.com/Manual/class-AudioListener.html

[10] https://docs.unity3d.com/Manual/class-SpriteRenderer.html

[11] https://docs.unity3d.com/Manual/class-Rigidbody2D.html

[12] https://docs.unity3d.com/Manual/Prefabs.html

[13] https://docs.unity3d.com/Manual/Collider2D.html

[14] https://docs.unity3d.com/Manual/class-Transform.html

[15] https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-Physics2DRaycaster.html

[16] https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-Canvas.html

[17] https://docs.unity3d.com/Manual/class-Tilemap.html

[25] https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html

InGame Tilemap Editing - 2D Level Editor with Unity - Tutorial

[18] https://www.youtube.com/watch?v=hx3ieBzVNg0

[19] https://github.com/MichelleFuchs/InGameTilemapEditing_Unity/blob/master/Episode_1/Assets/Scripts/Singleton.cs

How to make a Level Editor in Unity
[20] https://www.youtube.com/watch?v=eWBDuEWUOwc

W3Schools JSON Introduction
[21] https://www.w3schools.com/js/js_json_intro.asp

Jsonhelper Stackoweflow
[22] https://stackoverflow.com/questions/36239705/serialize-and-deserialize-json-and-json-array-in-unity

Audió fájlok forrása Storyblocks
[23] https://www.storyblocks.com/

Inno Setup Compiler hivatalos weboldal
[24] https://jrsoftware.org/isinfo.php

Script forrása
[25] https://stackoverflow.com/questions/35219952/constant-for-appdata-locallow-in-inno-setup

7. Nyilatkozat

Alulírott Papp Márton, Gazdaságinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Képfeldolgozás és Számítógépes Grafika Tanszékén készítettem,

Gazdaságinformatikus BSC diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

2022.11.16, Papp Márton