Fox shooter

17.12.2020

Készitette:
Kolumbán Attila
Szász Csaba

Tartalomjegyzék

Bevezető

A videójátékok napjainkban a szórakoztatási ipar egyik legjövedelmezőbb ága, szinte minden AAA játék mögött legalább 30-40 millió dolláros büdzsé van, és a játékokat a reklámok alapján vásárolják az emberek. Legnagyobb videojáték fejlesztő stúdiók telerakják termékeik szerencsejátékkal, mikrotranzakcióval, kiegészítőkkel, ezzel fizetésre kényszerítve a játékost.

Ez 10 évvel ezelőtt másképp volt. Közösségekre voltak bízva egész játékok multiplayer oldalai, játékosok készíthették saját maguknak a tartalmakat, talán a legjobb példa erre a Counter Strike. Valahol itt váltak nagyon népszerűvé a flash játékok, hála az Adobe Flash Playernek, bárki készíthetett szinte bármilyen játékot egyszerűen, és hamar. Mindenki játszott hasonló, egyszerű, de mégis szórakoztató játékokkal élete során, ahol tudta, hogy a játékot nem a pénzért készítették, hanem szeretetből.

A mi projektünk célja egy hasonló kis játék elkészítése volt, mejnek mechanikái egyszerűek, játékmenete gyors, és kis intervallumokban is játszható.

Célkitűzések

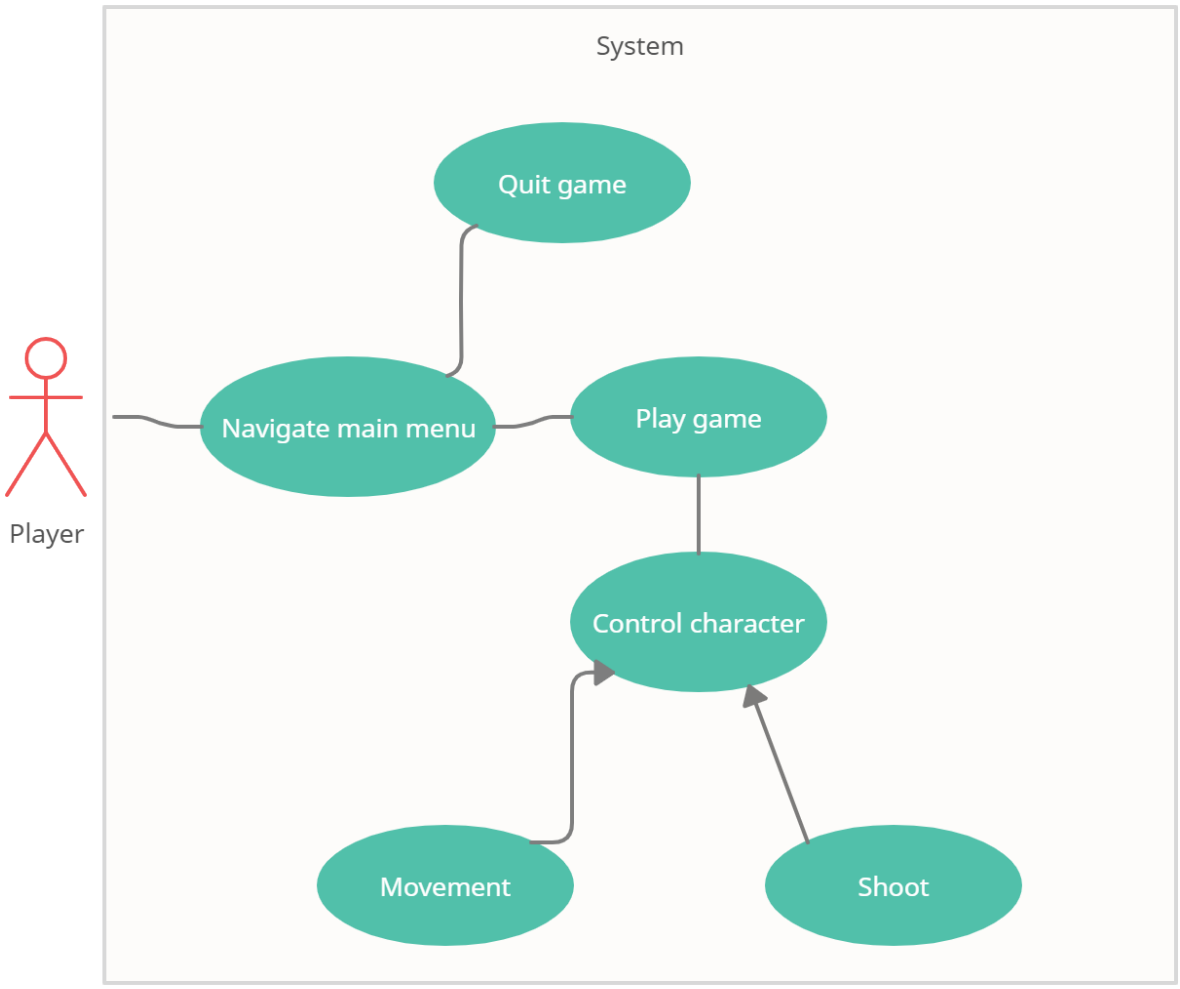
A projektünk célja egy gyors és egyszerű, mindenki által játszható 2d shooter/platformer játék elkészítése volt, a következő funkciókkal:

1. Mozgatható játékos karakter
2. Elsüthető és célozható fegyver a játékos számára
3. Ellenséges AI, amely folyamatosan támad
4. Pályán átugrandó akadályok
5. Játék menü rendszer
6. Hangefektusok

Követelmények

Felhasználói követelmények

1. A felhasználó tudja elindítani a játékot
2. Tudjon a játék menüjében navigálni



use-case diagram

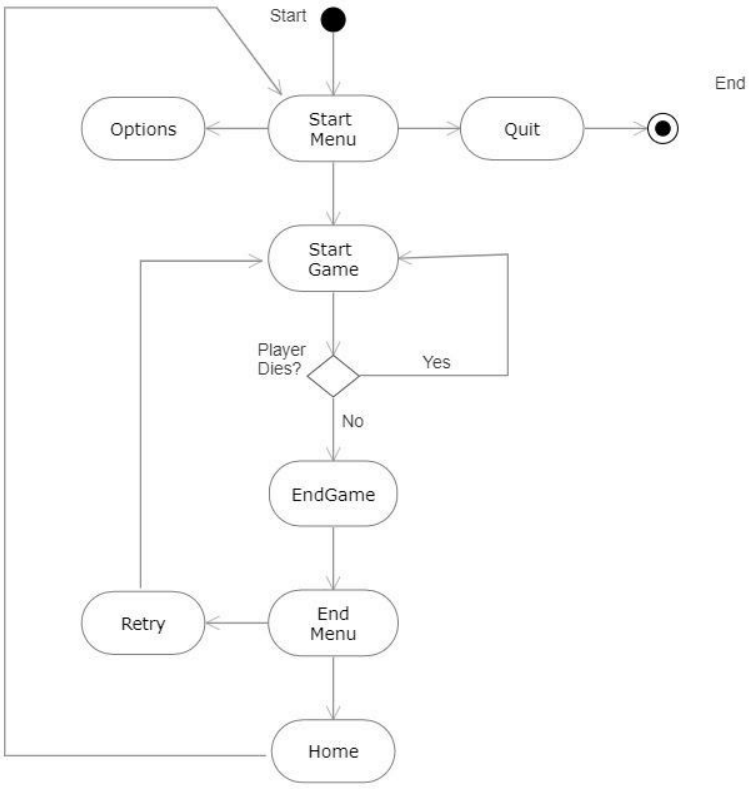
Rendszerkövetelmények

Funkcionális:

1. A játékos által vezérelt karakter kell tudjon mozogni és el kell tudjon jutni a pálya végére
2. A játékos kell tudjon lőni
3. Legyenek ellenfelek, amelyeket le lehet lőni
4. Az ellenfelek támadjanak a játékosra
5. A menüben lehessen navigálni

Nem-funkcionális:

1. Windows 10 - 64 bites rendszer
2. 50Mb merevlemez terület
3. DX10, DX11, DX12 capable



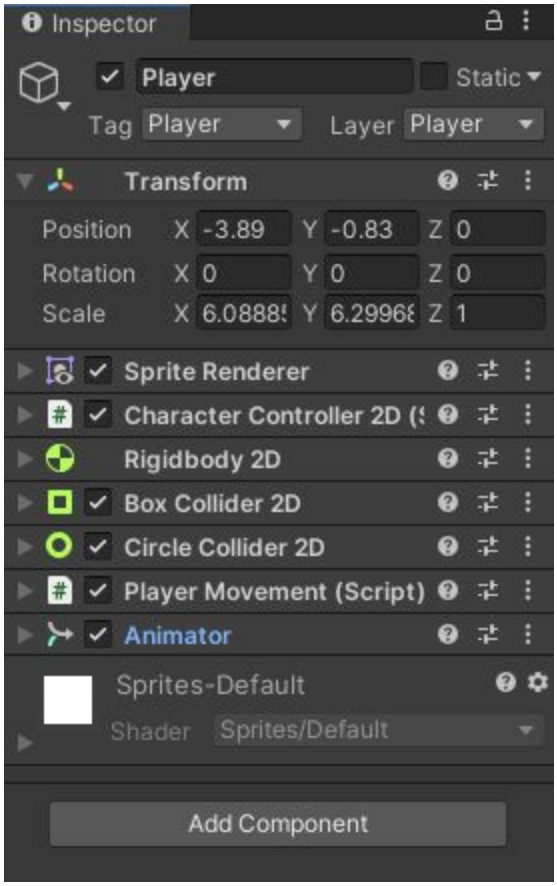
Működési elv

Unityben minden object különböző komponensekből áll, ezekkel határozzuk működésük és viselkedésük.

A játék középpontjában a Player áll, e köré építettük fel a projektet.

A játék 3 sceneből áll, ezek közül a legfontosabb a játék sceneje, ami tartalmazza az ellenfeleket, a pályát, a játékost, és ezek funkcionalitásait.

Player



Itt láthatjuk a különböző komponenseket, melyek alkotják a playert.

A két collider felelős a hit-detection miatt, az Animator komponens kezeli az animációkat,

A RigidBody ad fizikát a játékosnak, a Character Controller illete a Player Movement pedig lehetővé teszi a játékos mozgatását billentyűzettel.

A Player objektnek van 5 darab child objektje, ezek szükségesek különböző funkcionalitások megvalósítására.

A CeilingCheck és GroundCheck felelős meghatározni, hogy a karakter jelenleg a földön helyeszkedik el, illetve hogy tud-e gugolni, vagy ugrani.

A Main Camera a nézet, amit a játékos lát, ez követi a játékost.

A CharacterController2D script egy ingyenesen használható szkript, ami a játékos mozgásának alapjait határozza meg.

A Player Movement szkriptben határozzuk meg, mi történik adott gombok lenyomásakor

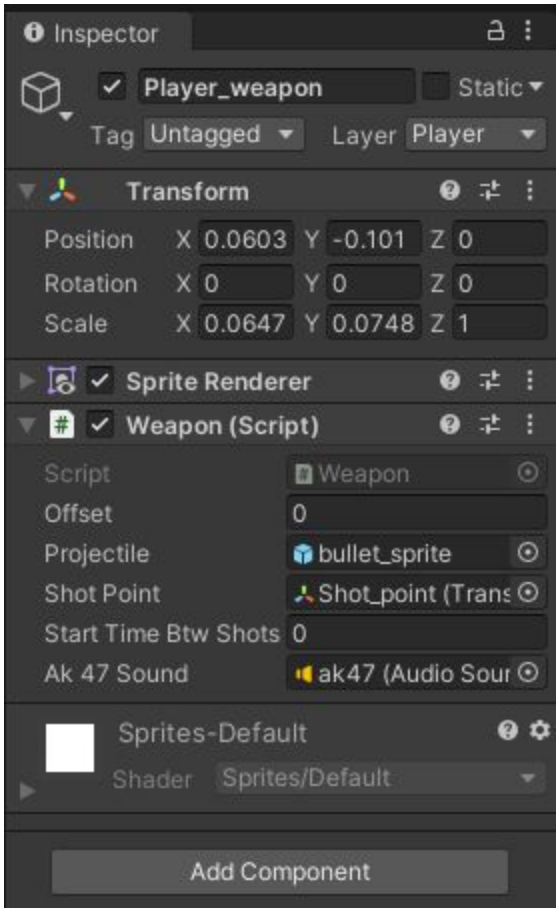
```
void Update()
{
    horizontalMove = Input.GetAxisRaw("Horizontal") * runSpeed;


    animator.SetFloat("player_speed", Mathf.Abs(horizontalMove));

    if(Input.GetButtonDown("Jump"))
    {
        jump=true;
        animator.SetBool("player_is_jumping",true);
    }

    if(Input.GetButtonDown("Crouch"))
    {
        crouch=true;
    }else if(Input.GetButtonUp("Crouch"))
    {
        crouch = false;
    }


}
```

Weapon

A Weapon object child objektje a Playernek, ez azt jelenti, hogy örökli annak helyét, és orientációját.

Egy child objektje van a Weapon objektnek, a ShotPoint, ez határozza meg, hol hagyja el a lövedék a fegyvert.

A fegyvernek fontos, hogy mindig a kurzor fele nézzen, így a lövések célzottan leadhatóak, illetve ha a játékos a saját háta mögé céloz, akkor forduljon meg. Ezeket a funkcionalitásait a Weapon szkriptben határozzuk meg.

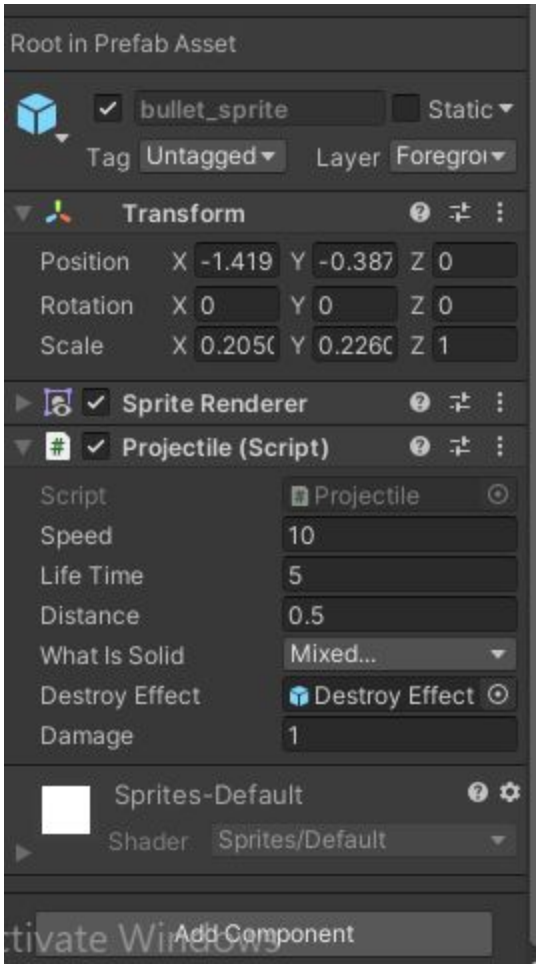

Inspector
Player_weapon
Static
Tag Untagged
Layer Player
Transform
Position X 0.0603 Y -0.101 Z 0
Rotation X 0 Y 0 Z 0
Scale X 0.0647 Y 0.0748 Z 1
Sprite Renderer
Weapon (Script)
Script Weapon
Offset 0
Projectile bullet_sprite
Shot Point Shot_point (Trans
Start Time Btw Shots 0
Ak 47 Sound ak47 (Audio Sour
Sprites-Default
Shader Sprites/Default
Add Component

```
void Update()
{
    Vector3 difference = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;
    float rotZ = Mathf.Atan2(difference.y, difference.x) * Mathf.Rad2Deg;

    //weapon rotation limitation and flipping
    //Debug.Log(rotZ);

    if(rotZ > 90 || rotZ < -90)
    {
        //rotZ = 120;
        //transform.rotation = Quaternion.Euler(180f,0f,0f);
        mySpriteRenderer.flipY = true;
    }
    else
    {
        //rotZ = -120;
        mySpriteRenderer.flipY = false;
    }
    //----------------

    transform.rotation = Quaternion.Euler(0f, 0f, rotZ + offset);

    if (timeBtwShots <= 0)
    {
        if (Input.GetMouseButtonDown(0))
        {
            ak47Sound.Play();


            Instantiate(projectile, shotPoint.position, transform.rotation);
            timeBtwShots = startTimeBtwShots;
        }
    }
    else
    {
        timeBtwShots -= Time.deltaTime;
    }

}
```

Ebben a szkriptben tesszük lehetővé a fegyver kurzor irányába való fordulását, illetve gomb nyomására lövedék létrehozását.

Projectile

A bal egérgomb lenyomása esetén, a játék a Fegyver csövén elhelyezkedő ShotPoint objekt helyére elhelyez egy lövedéket, amely a kurzor irányába halad, és adott dolgokkal való ütközés esetén felrobban, illetve ha kell, sebzi az ellenséget.




```
void Update()
{
    RaycastHit2D hitInfo = Physics2D.Raycast(transform.position, transform.right, distance, whatIsSolid);


    if(hitInfo.collider != null)
    {
        if (hitInfo.collider.CompareTag("Enemy"))
        {

            hitInfo.collider.GetComponent<Enemy>().TakeDamage(damage);
        }
        DestroyProjectile();
    }

    transform.Translate(Vector2.right * speed * Time.deltaTime);
}

void DestroyProjectile()
{
    Instantiate(destroyEffect, transform.position, Quaternion.identity);
    Destroy(gameObject);
}
```

A Projectile szkriptben látható, hogy ha a lövedék egy ellenségnek ütközik, megsemmisítődik, megsebzi az ellenséget amit eltalált, illetve létrehoz egy részecske effektust.

Mielőtt még az ellenségekről beszélnénk, szükséges bemutatni egy komponenst, ami lehetővé teszi a Unity projektekben Pathfinding algoritmusok egyszerű alkalmazását.

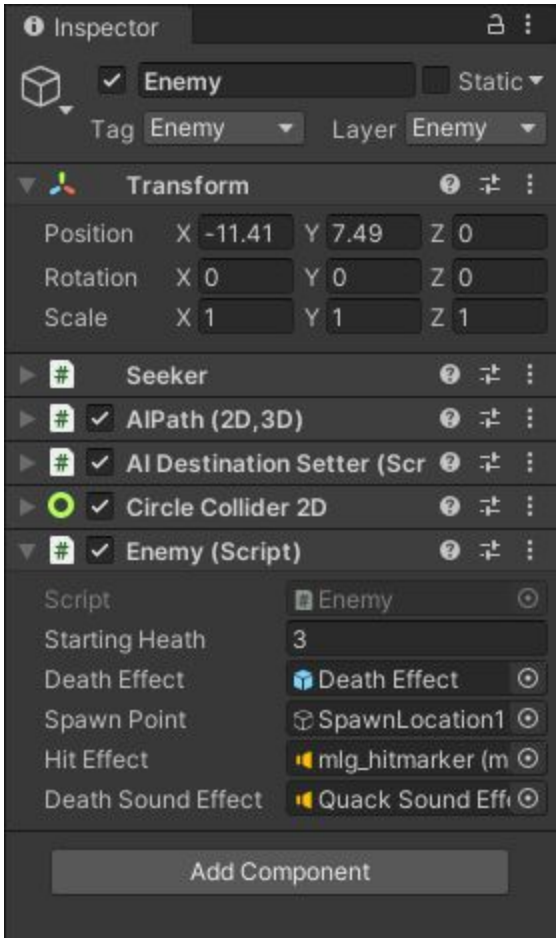
A* Pathfinding project

Az A* Pathfinding egy ingyenesen, illetve opcionális fizetősen használható Unity package.

Importálása után előre elkészített szkriptjei segítségével gyorsan és egyszerűen alkalmazhatunk fejlett és optimális útkereső algoritmusokat játékunkban.

Mi a projektünkben Grid Graph-ot használtunk, mivel működése egyszerű és gyors.

Enemy



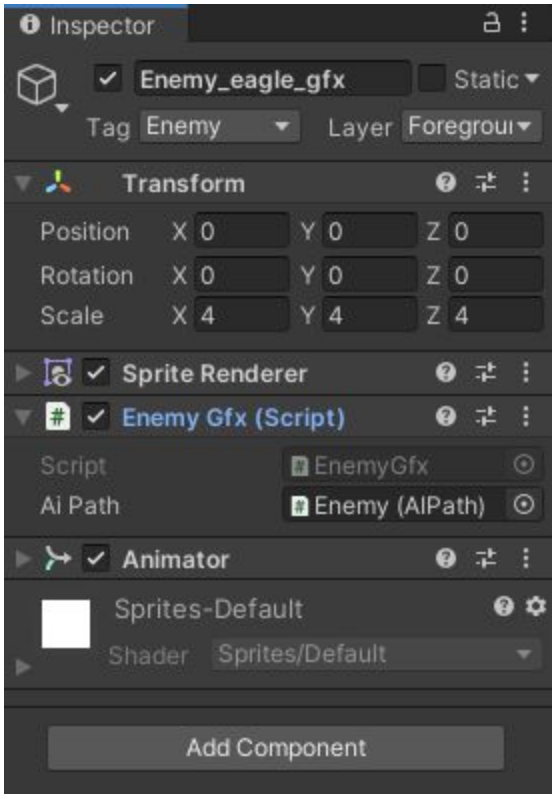
Az enemy objektünknek szükséges a játékos fele közeledni, ezzel veszélyeztetve őt.

Ütközés esetén a játék újraindul. A Seeker, AIPath, AI Destination Setter A* által biztosított szkriptek, ezek kezelik a Grid Graphot, megkeresik a lehető legrövidebb utat a játékos fele, és mozgatják azt.

```
void Update()
{
    if(health <= 0)
    {
        deathSoundEffect.Play();
        Instantiate(deathEffect, transform.position, Quaternion.identity);
        //Destroy(gameObject);
        transform.position = spawnPoint.transform.position;
        health = startingHeath;
    }
}


public void TakeDamage(int damage)
{
    hitEffect.Play();
    health -= damage;
}

private void OnTriggerEnter2D(Collider2D collison)
{
    if ( collison.gameObject.tag == "Player")
    {
        SceneManager.LoadScene( SceneManager.GetActiveScene().name );
    }
}
```

Az Enemy szkript tartalmazza a logikát, amely alapján az ellenség ha elszenved 3 találatot, megsemmisül, illetve elteleportál egy a játékostól távoli helyre, újra megtámadva azt, ezzel biztosítva a folyamatos nyomást a játékoson, és a konstans nehézséget.

Az Enemy objektnek egy Child Objektje van, amely tartalmazza az animációját, illetve egy szkriptet annak forgatására, ha szükséges




```
// Update is called once per frame
void Update()
{
    if(aiPath.desiredVelocity.x >= 0.01f)
    {
        transform.localScale = new Vector3(-4f, 4f, 4f);
    }
    else if(aiPath.desiredVelocity.x <= -0.01f)
    {
        transform.localScale = new Vector3(4f, 4f, 4f);
    }
}
```

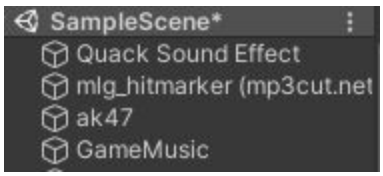
Ez csak egy egyszerű szkript, amely mindig az animációt abba az irányba fordítja, amelybe épp halad az ellenfél.

Character fall hitbox

A pálya alatt elhelyezkedik végig egy hitbox, ezzel a játékos csak akkor tud érintkezni, ha elrontott egy ugrást, tehát ebben az esetben a játék újraindul. Egy egyszerű szkript elég ennek a megvalósítására.

```
public class CharacterFall : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collison)
    {
        if ( collison.gameObject.tag == "Player")
        {
            SceneManager.LoadScene( SceneManager.GetActiveScene().name );
        }
    }
}
```

Hangefektusok



A Scenen elhelyezkedik még illetve az összes hangefektus amelyet játékunk felhasznál.

Ezeket különböző szkripteken belül hívunk meg, amelyek láthatóak az előző képeken.

Események amik során hangefektusok kerülnek lejátszásra: játék indítása, fegyver elsülése, ellenség eltalálása, ellenség elpusztulása.

Grid

A pálya felépítését egy 3 grid segítségével építettül fel. Ezek a gridek könnyebbé teszik a pálya grafikus elemeinek rajzolását. 3 szint van, a background, amely konkrétan az ég, ez picitt lassabban mozog mint a többi layer, egy 3d effektus előidézése érdekében.

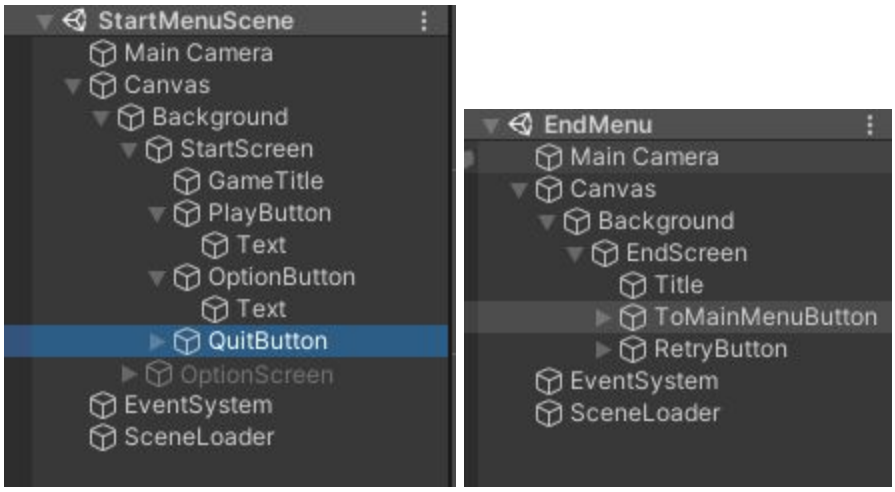
A Middleground, amely különböző objekteket tartalmaz, mint például falak, fák, amik csak dekorációs célt szolgálnak.

A Foreground, amely azon elemeket tartalmazza , amik szolidak, és a játékos használhatja őket mozgásra.

Scenes

Az eddig felsorolt objectek, mind a játék scene részei, itt tartózkodik a legtöbbet a játékos.

A main menü, meg az end game menü két másik scene, amelyről a játékos elérheti a játékot, vagy bezárhatja azt.



Repository management

A repositoriban minden nagyobb featuret külön ágon hoztunk létre, majd miután kész volt, tesztelve volt, és működött, mergeltük a main ággal.

Hogy lássuk ki mivel dolgozik éppen, használtuk a github Project bordját, ahol feltüntettük mely részek vannak már készen, mik vannak még hátra, és éppen ki micsodával dolgozik.

Tesztelés

A projekt készítése során nagyon fontos annak tesztelése. Mivel egy játékról van szó, a legfontosabb tényező a játékos szórakoztatása. Ennek érdekében mi is, illetve több barátunk is játszottunk a játékkal különböző állapotokban, és visszajelzés függően alakítottuk azt. Erre példa a játékos sebessége, az ugrás erőssége, illetve az ellenfelek mennyisége. Ezek mind olyan dolgok, amelyek csak tesztelés során derülnek ki, hogy működnek vagy nem. A végtermékkel meg vagyunk elégedve, terv szerinti hangulatot ad át a játék.

Összefoglalás és Következtetések

A projekt kitűzött céljait elértük. Megvalósítottunk egy retro egyszerű shooter/platformert amely szinte bárki által játszható és élvezhető. Lehetőség van ennek továbbfejlesztésére,

például logikus fejlesztések egy többjátékos mód, több fajta fegyver, több fajta pálya, illetve több fajta ellenség.

Mellékletek

https://arongranberg.com/astar/features#

https://github.com/Brackeys/2D-Character-Controller

https://www.youtube.com/user/Brackeys

https://assetstore.unity.com/packages/2d/characters/sunny-land-103349

https://assetstore.unity.com/packages/2d/gui/icons/ultimate-2d-weapons-warfare-7-weapons-134023