Projektmunka Project_X

Hencz Kornél
Ozvald Kornél

Fejlesztői dokumentáció

A projektmunkát két fős csoportban végezte el Ozvald Kornél és Hencz Kornél.

Rövid leírás

A játékot két játékos játssza. Feladatuk a „Sandox” térkép felderítése, a különböző aktivitás pontok megtalálása, és interakcióba lépés velük. A legjellemzőbb aktvitás pontok egy másik kétdimenziós platform világba „viszik” a játékosokat, ahol a lehető legjobb tudásuk szerint kell egymást átsegíteniük az akadályokon.

Technikai bevezető

A szoftver Unity játék motor alatt fut, annak beépített moduljaival, illetve jelenleg egy külső fejlesztői csomag felhasználásával készültek további program megoldások. Ez a Photon Engine nevű Unitybe implementálható, C# nyelven programozható plugin, amely különböző hálózati megoldásokkal segíti a többjátékosra szabott fejlesztést.

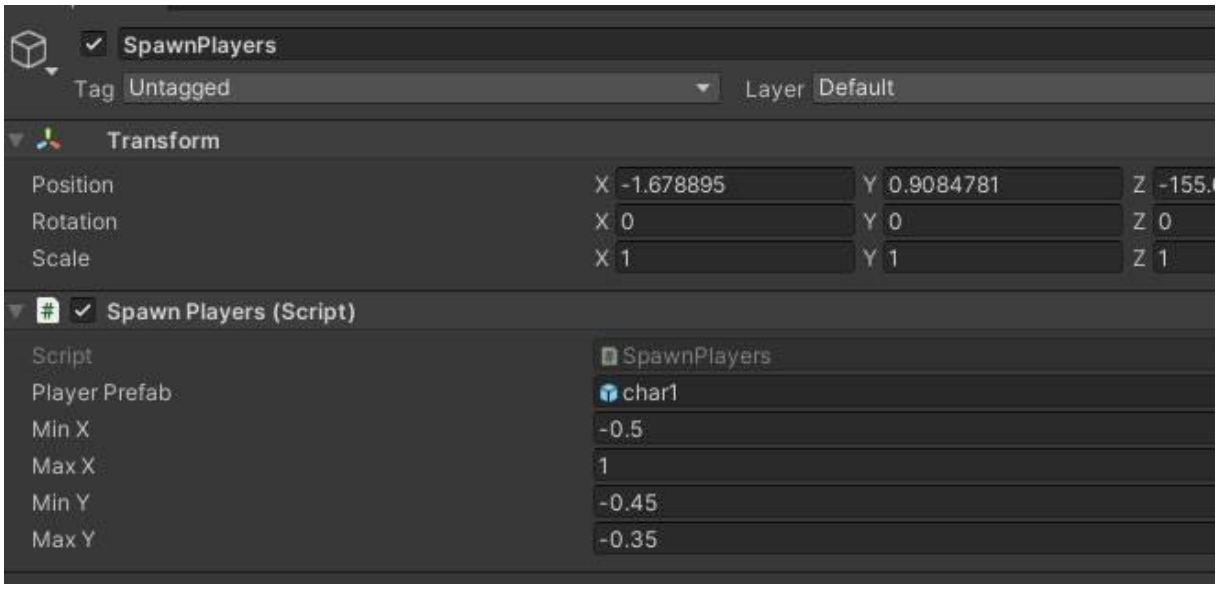
Fejlesztés menete

Karakter, 2D

A SandBox világ elve, hogy a karakter kétdimenzióban létezik. Ez azt jelenti, hogy az elérhető, létező játék terület két tengelyen oszlik fel egy X, és egy Y-ra. A modellezett világ egyes lokációs pontjai tehát (X,Y) alakban írhatóak fel. A játékos karakteréhez pedig szintén ez a legjellemzőbb hozzá társított érték, ami azt reprezentálja, hogy logikailag a térkép éppen melyik pontján lelhető fel, rajzolandó ki.

A játékmenethez való csatlakozással egy bizonyos kereten belül előállított véletlenszerű X, és Y értékkel kirajzolásra kerül a karakter a hozzá tartozó koordináta pontban. Ezzel gyakorlatban elérve a játékos „létezését”. Technikailag ez úgy néz ki, hogy a SandBox világ egy Unity Sceneként van megvalósítva, és egy ebben jelenlévő játék objektum, a „SpawnPlayer”, és a

hozzá tartozó játék script példányosíja a játékost, a hozzá tartozó scriptekkel, és a hozzátartozó textúra elemmel(sprite).




```
Vector2 randomPosition = new Vector2(Random.Range(minX, maxX), Random.Range(minY, maxY));
PhotonNetwork.Instantiate(playerPrefab.name, randomPosition, Quaternion.identity);
```

A PhotonNetwork alapú példányosításról később..

Természetesen a játékmenet egyik legalapvetőbb eleme, hogy a felhasználó a karakterének pozícióját módosíthatja, „járhat" vele. Ennek háttere az egyed grafikus, térbeli transzformálása, valamilyen esemény hatására. A játékmentben ez egy direkt tevékenység, a játékos kérésére megy végbe. A játékos karakter példányhoz tartozik egy script ami ezért felelős. Az update metódusa folyamatosan bementet ellenőríz, ami horizontális, és vertikális tengelyekre vannak kihatással. (Fel-le(Y), jobbra-balra(X)). Amennyiben érkezett valamelyik vizsgált inputból, akkor abból egy vektort állít elő, és az alapján elvégzi a grafikus megjelentésben a módosítást. Ezzel a karakterhez tartozó koordinációs metaadat is módosul.

```
float x = Input.GetAxisRaw("Horizontal");
float y = Input.GetAxisRaw("Vertical");

moveDelta = new Vector3(x,y,0);

// Scale állításával tükrözöm a karaktert, ezzel elérve az hatást a fordulásnak.
if(moveDelta.x > 0){
    transform.localScale = Vector3.one;
}
else if(moveDelta.x < 0){
    transform.localScale = new Vector3(-1,1,1);
}
```

Collision HIT Rendszer

Játékélmény szempontjából nem kedvező, ha nincsenek a játékos körül fizikai korlátok. Egyes pontok a térképen elérhetetlenek, hiszen ott valamilyen objektum van, ami elzárja a teret a bejárás lehetőségétől, például egy fa. Ezt a játékfejlesztésben Collision rendszernek hívják. Valamilyen hierarchia szerint priorizáljuk a grafikus rétegek „felsőbbrendűségét", ez annyit jelent, hogy ha L1 (layer1) réteg L2 felett van, akkor grafikusan is az L1-hez tartozó textura takarni fogja az L2-höz tartozót. Ez a rétegezés tranzitív, lineárisan csökken a „takaró képesség" a hierarchiában való lefele haladáskor. Tehát ha L1 takarja L2 réteget, és L2 réteg takarja L3 réteget, akkor L1 takarja L3-at is.

Abban az esteben viszont, ha a játékoshoz tartozó karakter objektum grafikája azonos rétegben van egy másik objektummal (pl másik játékos, fal ..) tehát nem egyértelmű, hogy eltűnik

```
transform.Translate(moveDelta.x * Time.deltaTime * fixedCol(moveDelta, "x"), 0, 0);
```

```
private int speed = 5;
```

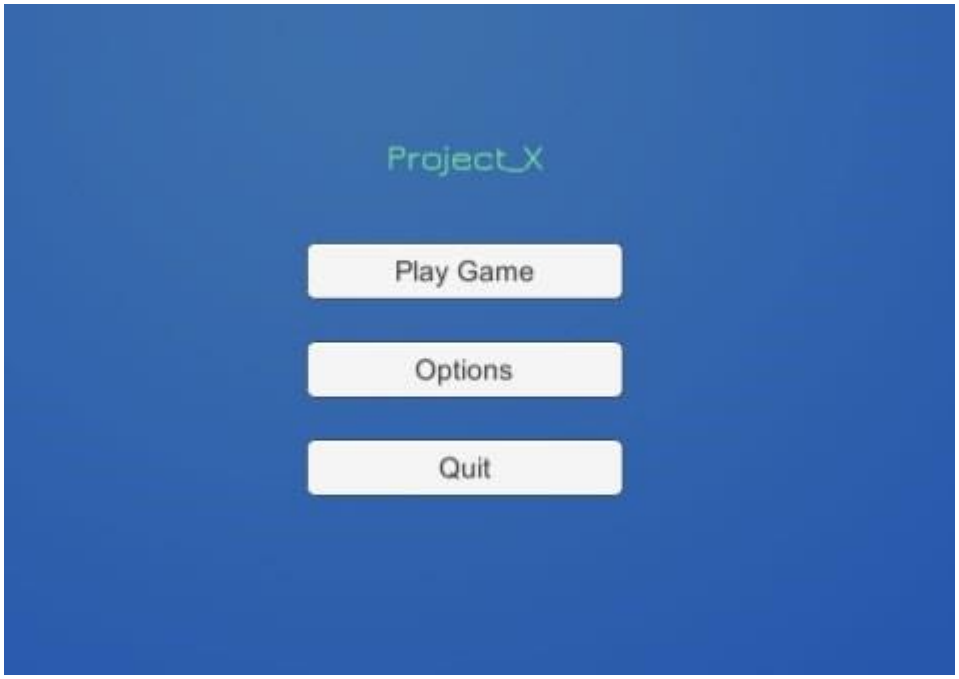
```
if(way == "y"){
    for(int i = 1; i <= 5; i++){
        if(Physics2D.BoxCast(transform.position, boxCollider.size, 0, new Vector2(0,moveDelta.y),
            return i-1;
        }
    }
}
if(way == "x"){
    for(int i = 1; i <= 5; i++){
        if(Physics2D.BoxCast(transform.position, boxCollider.size, 0, new Vector2(moveDelta.x,0),
            return i-1;
        }
    }
}
return speed;
```

mögötte, vagy átmegy felette, akkor meg kell akadályozzuk, hogy az adott X, Y pont elérhető legyen a számára.

Ennek megoldásában a Unitybe implementált BoxCollider2D kiegészítő nagyszerű lehetőséget nyújt. Adott objektumra való alkalmazása esetén olyan extra egyed mezők, és metódusok válnak elérhetővé, amikkel dolgozva a kivitelezés is fejlesztőbarát marad. Felvehetünk a textúra környékén több pontból álló síkidomot, amit aztán osztályozhatunk, besorolhatunk, elnevezhetünk végeredményében egy olyan pont halmazt kapunk, amibe egy másik dedikált ponthalmaz beleérése, vagy érintkezése vele meghív valamilyen eseményt.

Esetünkben a BoxCollider halmazok, mivel csak a játékos környezetében elérhető interakciókra van kihatással, ezért a Player Scripben lévő mozgásrendszerbe van implementálva a Collision területek vizsgálása.

Menü rendszer



A menü három darab Unity Sceneből épül fel egy Main, egy Loading(Connecting), és egy RoomManagaerből(Join).

Az egyes scenek közötti váltást a Unity SceneManagement modul valósítja meg.

```
SceneManager.LoadScene("LoadingScene");   using UnityEngine.SceneManagement;
```

A Menü Gombokhoz „Button Click” események vannak hozzá rendelve. Az egyes menüpontok egyenként más Scenet töltenek be.

- Main Menü Scene: Három funkció gombot tartalmaz, egy „play game”, egy „options”, és egy „quit” található meg itt. Az Options lehetőség egyelőre nem rejt beállítással kapcsolatos extra lehetőséget, a Quit csupán szofisztikált megoldást nyújt a szoftverből való kilépéshez, a leállításához. Érdekesebb a Play Game lehetőség, amit egy Loading Scene követ
- Loading Scene(Connecting): A szoftver itt tesz kísérlet kapcsolatot létesíteni a kiszolgáló (PHOTON) szerverrel, amennyiben ez sikertelen a főmenübe való visszakerülés fog megtörténni, ebben az esetben vagy a felhasználó hálózatával van probléma, vagy a kiszolgáló állt le. Normális esteben a felhasználó átkerül a Lobby, az az RoomManager Scenbe.
- RoomManager Scene: Három további opció, a megfelelő tetxbox kitöltésével játék létrehozása (mint kiszolgáló), vagy meglévő játékhoz csatalakozás (mint normális klines), vagy vissza lépés a főmenübe, ami a szerverről történő lecsatlakozással jár.

<u>Menü elemek technikai felépülése:</u>

Egy Canvas unity objektum szolgál alapul, amire panel típusú rögzíthető, mint gyermek, és a gyermekre jellemző érték a BackGround image, ennek konfigurálása adja a menü háttér küllemét. A panel mellett még megtalálhatók gomb(button) típusú objektumok is, aminek property paramétereinek igény szerinti konfigurálása jellemzi külsőjét, és az onclick eseményekhez társítandóak a különböző menüvel kapcsolatos eljárások.

Hálózati megoldás

A hálózati megvalósítás hátterét a PHOTON motorként ismert csomag biztosítja. Online szervert biztosít olyan beépített funkciókkal, mint saját objektum kezelés, amit sűrűn ki tud szórni a klienseknek, vagy a fejlesztő által manuálisan implementálható esemény trigger pontok, és az azokhoz tartozó eljárások.

A kézfogás úgy történik meg, hogy a fejlesztő az applikációjához ingyen token kódot kap, ami egy szerverre mutat rá. A felhasználók az szoftverhez való csatlakozáskor a tokenhez kapcsolódó szerverrel elkezdenek kommunikálni TCP/IP protokollal. Amennyiben a csatlakozáshoz szükséges feltételek teljesülnek, a fő „poolon" belül, különböző alkiszolgálókat, szobákat hozhatnak létre, vagy csatlakozhatnak meglévőhöz.

```
public void NewRoom()
{
    PhotonNetwork.CreateRoom(CreateRoomID.text);
}
```

```
public void JoinRoom()
{
    Debug.Log(JoinRoomID.text);
    PhotonNetwork.JoinRoom(JoinRoomID.text);
}
```

A CreateRoom eljárás végrehajtása a klienset Master klienssé fokozza(szerverré), és az OnJoined room lokális metódus triggerelődik.

```
public override void OnJoinedRoom()
{
    PhotonNetwork.LoadLevel("SandBoxScene");
}
```

A PHOTON motor általi SceneMenedzser a klienseknek szolgáltatja a hálózatban példányosított objektumok listáját, és a velük végzendő feladatokat.

```
PhotonNetwork.Instantiate(playerPrefab.name, randomPosition, Quaternion.identity);
```

A PHOTON példány létrehozó térben képzi első sorban az objektumot, hisz ezekre általában jellemző grafikus létezés is, ezek nem scipt fájlok.

Hálózati játékmenet (Alfa verzió):

1. Játékos csatlakozik Master vagy Kliensként (Játékos objektum létrejön)
2. Viewot rendel hozzá, amiről az Update metódus folyamatos jelentést kér, és a jelentésben vissza érkező adatokkal történik a kiértékelés, munka végzés (objektum grafikai transzformáció például.)

```
PhotonNetwork.Instantiate
```

SandBoxScene:

A main mappunk a SandBox map ahol található egy mocsár, egy erdő, a tengerpart és egy kisebb falu, ami tartalmazza az NPC-ket, ahonnan majd küldetéseket indíthatunk el. A scene sokféle

tile-ból és sprite-ból épül fel ezzel megpróbálva hangulatossá tenni a környezetet. A collision-k megoldására 2D tilemap collider lett használva nagyrészt, valamint egyes elemeken/területeken polygon colliderek is. Ebben a scene.ben található még egy ágy valamint egy lootbox, amikhez még jelenleg nem társul funkció.

Kamera:

A karakterekek követését Cinemachine segítségével oldottuk meg. A karakterek létrehozásakor létrehozunk egy hozzá tartozó virtuális kamerát, ami követni fogja a karaktert. A cinemachine segítségével könnyen állítható a karaktert követő kamera viselkedése, mint például a deadzone-k beállítása, vagy a kamera reakciója a karakter elmozdulására.

```
public void DisplayNextSentence(){
    if (sentences.Count == 0) {
        EndDialogue();
        return;
    }
    string sentence = sentences.Dequeue();
    StopAllCoroutines();
    StartCoroutine(TypeSentence(sentence));
    }
```

1Mondatok kiiratása

NPC interakció:

Amikor a játékos találkozik egy NPC-vel a játékban, és arra rákattint akkor megjelenik egy animált dialogueBox, az NPC nevével és mondandójával. A space billentyű lenyomásával tovább léptetheti a szöveget a játékos, majd a beszélgetés után, a platformer szintet tölti be a játék.

Platformer level:

A játékosok az NPC-ket követő interakciók után, egy új scene-be töltenek be, ez a platformer

```
private void OnTriggerEnter2D(Collider2D collision) {
    if (collision.gameObject == LoadPlayers.GetPlayer()){
        PhotonNetwork.LoadLevel("Level1");
    }
```

2Collision Detection

map. Ez a Scene több elemből épül fel, köztük egy Parallax effektet használó háttérből, egy tilemap háttér- és előtérből, valamint a fő tilemap-ből. Ezen a fő tilemappen található egy 2D tilemap collider ami lehetővé teszi, hogy a karakterünk ne essen keresztül a textúrán. Ezen kívűl található egy "Spikes" nevű tilemap, colliderekkel, amire elhelyeztünk egy scriptet, mellyel való érintekzéskor a játékos karaktere a pálya elejére kerül.

A játékosokra a szint elején rákerül egy 2D rigidbody component, hogy szimuláljuk a gravitációt. Abban az esetben ha sikerül a platformer pályán végigmenni, akkor a pálya végén található ajtónál visszatöltünk a "SandBox" scene-be.

```
private void OnTriggerEnter2D(Collider2D collision) {
    GameObject collisionGameObject = collision.gameObject;
    if (collisionGameObject.gameObject == LoadPlayers.GetPlayer()) {
        PhotonNetwork.LoadLevel("SandBoxScene");
    }
```

3Visszatöltés a SandBox világba

Verziók

I. Pre_Alpha 0.1: SandBox Word Skála, "Texture Atlas" implementálás
   a. 0.2: Alap world buildup
   b. 0.3: Karakter script implementálás(kezdeti)
   c. Collision Hit script
II. Alpha 1.0: Komplett SandBox pálya, krakter script, mozgás véglegesítés.
   a. 1.1: Pálya fel "layerezés"-e
   b. 1.12: Pálya fel "collision buildelése"
III. Alpha 1.2: Hálózati motor implementálás(Photon)
   a. 1.21 Karakter script hálózati megvalósítása.
IV. Alpha 1.3: Menü rendszer, lobby implementálás
V. Alpha 1.4: NPC interakció
   a. 1.41: Platform világ implementálás, scene átváltás.
VI. Beta 0.1: Bugfix, prezentálásra felkészítés. Struktúrált továbbfejlesztésre kész.

Továbbfejlesztési lehetőségek

A játékot még sokféleképpen lehet majd a jövőben továbbfejleszteni, mint például háttérzene hozzáadásával, hang- és részecske effektek létrehozásával. Ezek mellet a jelenleg tartalmazott részek további finom hangolásával (QoL Changes), új pályák tervezésével és természetesen felhasználói visszajelzés segítségével.

Hencz Kornél UNPRMW
Ozvald Kornél HAY7SJ
OE-AMK