1. Unreal Engine

Az unreal engine egy játék engine, amit az Epic Games fejlesztett ki. 1998-ban volt az első demójuk, egy lövöldözős játék, melynek az volt a neve, hogy Unreal elöször csak egy PC (ezen belül is a játékokra volt tervezve az engine) volt a cél platform, de azóta a filmiparban is jelentős szerepet játszott. Az unreal engine egy C++ írt szoftver a többi játék enginenel szemben az unreal engine már a kezdetekben is használt vizulális scripting nyelvt a magasszintű játék funkciók létrehozásához Unreal Engine 4 elött ezt UnrealScript-nek hivták, majd ezt lecserélték Blueprint-re.

1.1. UnrealScript

Egy objektum orientált nyelv volt, ami hasonlít a java-ra, egyszres öröklődés volt benne, és nem volt burkoló osztály primitív típusokhoz. Az interface-eket csak az unreal engine 3-tól támogatott, támogatja az operátor túlterhelést, de metódus túlterhelést nem.

1.2. Blueprints

Egy vizuális scripting nyelv teljes játéklogikákat lehet gyorsan és hatékonyan leprogramozni egy node alapú interface segítségével, ez egy script nyelv, ami az engine-ben definiált objektum orientált osztályokat használ. a Blueprint rendszer rettentő hatékonnyá teszi a fejlesztést, mert azonkívül, hogy saját Blueprint osztályokat hozhatunk létre, ezt a rendszert nem csak programozok tudják használni, könnyen tanulható (pl.: grafikus csapat is tudja használni nemcsak a fejlesztők).

1.3. Blueprint fajták

A Blueprint-eknek több fajtája van, és mindegyiknak saját feladata van, “scripting level events” interface-ek definiálása vagy makrók definiálás.

1.3.1. Blueprint class

Gyakran Blueprint-nek vagy Bp-nek rövidítik. A tartalom gyártóknak megkönnyíti a munkáját és ők könnyen tudnak hozzáadni funkciókat a meglévő játéklogikához. A Blueprint-eket az Unreal Engine-en belül készítik vizuálisan.

1.3.2. Just-Data Blueprint

A Csak adat Blueprint az egy Blueprint osztály, amiben csak kód van változok örökölt komponensek, ezeket lehetőségünk van módosítani, de nem lehet új elemet hozzájuk adni.

Ezek olyanok, mintegy sablon, és a dizájnereknek lehetőséget ad, hogy a változók értékait finomítsák, amig meg nem kapják kívánt eredményt.

1.3.3. Level Blueprint

A „Level” Blueprint az egy olyan Blueprint, ami egy esemény grafikonként funkcionál. Minden szintnek vagy egy sajt Szint Blueprint-je ezek automatikusan létrejönnek, de újat nem lehet manuálisan létrehozni.

1.3.1. Blueprint Interface

A Bleprint interface az egy függvény név halmaz, ami nem tartalmaz implementációt, és ezeket a függvényeket tudjuk másik Blueprint osztályokhoz adni. Minden Blueprint amihez hozzáadjuk az interface-eket tartalmazni fogják az interface függvényeit és ekkor lehet funkcionalitást adni nekik. nem tudnak új változót létrehozni, grafikont módosítani, vagy új komponenst létrehozni.

1.3.2. Blueprint Macro Library

A Blueprint Macro Library egy konténer, ami képes tárolni gyakran használt code elemeket, és ez által fel gyorsítják a fejlesztést

1.4.Steam

A Steam egy tartalom kezelő és továbbító rendszer, rengetek szoftvert árul a digitális áruházában (Steam Store), nagyrést játékokat. Kényelmi funkciókat is tartalmaz. Saját DRM-je, ami lehetővé teszi, hogy ne a géphez legyen kötve az a megvett szoftver, hanem a felhasználói fiókhoz.

1.4.1. Steam-es Funkciók

- Steam Csevegés: üzenet vagy hívással elérheted ismerősidet.
- Steam Közvetítés: Élőben közvetítheted a játékot az ismerőseidnek.
- Steam Műhely: Létrehozható saját kinézet, vagy egyedi mod készítésre teremt éhetőséget.
- Elérhető mobilon is
- Könnyű vásárlás: több mint 100 fizetési mód, és 35 pénznem
- Játékvezérlők támogatása, újra konfigurálása

1.4.2. Steamworks

Steamworks-nek hívjak a Steam-es API-t, ami tartalmazza az eszközöket a fejlesztők számára, hogy teljesen kihasználják a Steam kliens képességeit, de ezek csak lehetőségek nem kötelező velük élni.

1.4.3. Steamworks Funkciók

- **Common Reistributables**: lehetőség installálni komponenseket pl.: Microsoft Visual C++ redistributables
- **Game Notification**: értesítések offline aszinkron multiplayer játékokhoz
- **Microtransactions (In-Game Purchases):** játékbeli tárgyakat lehet venni, ami valamilyen speciális bonuszt ad annak, aki megveszi
- **Multiplayer mod:** többjátékos mód, Steam Matchmaking API, Steam Game Servers API
- **Stats and Achivements:** játékos mérföldkövek, statisztikák
- **Enhanced Rich Presence:** marketing eszköz
- **Steam cloud:** távoli file tároló lehetőség

- **Steam input:** több mint 100 támogatott verérlő
- **Steam DRM:** digitális jog kezelő, ez egy wrapper, és kikényszríti a steam fútatását a játék/szoftver megnyitásakor
- **Steam Error Reporting:** egy error kezelő rendszer 10 ugyanolyan hiba után feltölti a cloud-ba
- **Steam HTML Surface**: lehetővé teszi a HTML alap aldalok megjelenítést
- **Steam Inventory Service:** egy a funkció készlet, ami lehetővé teszi**,** hogy játékok külön szerver fútatása nélkül folyamatos leltárt hozzunk létre
- **Steam Keys:** Steam-en milyen termékhez lehet kulcsokat generálni, amit majd a vásárlok tudnak érvényesíteni
- **Steam Leaderboards:** folyamatos és automaikusan rendezet ranglisták
- **features/music_player:** lehetőséget ad arra, hogy saját lejátszási listánkat importáljuk steam-re
- **Steam Overlay:** Steam-en keresztül indult játékoknál, lehet elérni ezt a overlay-t ami hasonlít a Steam kliensre
- **Steam Remote Play:** lehetőség a távoli játszásra pl.: TV-n, telefonon…
- **Steam Screenshots:** automatikusan támogatott a Steam overlay-n keresztül
- **Steam Voice:** egy a funkció készlet, ami lehetővé teszi, hogy tömörített vagy nem tömörített mikrofon hangot rögzítsünk
- **Steam Workshop:** lehetőséget ad a felhasználóknak saját készítésük tartalmat adjanak hozzá a játékhoz vagy más szoftverhez
- **Steam Video:** video Streaming lehetőség
- **User Authentication and Ownership:** lehetőségeket ad a felhasználok hitelesítésre
- **Valve Anti-Cheat (VAC) and Game Bans:** eszközöket ad a csalok elleni harchoz
- **Virtual Reality:** VR támogatás

2. The Witch's Pact

Egy túlélős játék, ahol az a cél, hogy túl kell élni x ideiig majd megölni a főnököt. A játék Roguelike elemeket is tartalmaz, lelkeket kell gyűjteni és azokból lehet majd fejlesztéseket venni a következő menetre.

2.1. Játékmenet

A játékos egy boszorkány karaktert irányít billentyűzet segítségével, és fejlesztéseket, új támadásokat szerez a játék folyamán. A cél, hogy túléje 20 percing majd megölje a főnököt, aztán vége a játéknak, vagy ha nem sikerül ezt teljesíteni akkor veszít a játékos. Végeredménytől függetlenül a felhasználó szerez lelkeket, amit majd elkölthet fejlesztésekre, amik a következő játékmenettől lesznek érvényesek.

2.2. Játék specifikáció

3. Projekt előkészítése

Létrehoztam egy Blueprint projektet, majd bekapcsoltam az „Online SubSystem” plugin-t.

A defaultEngine.ini-ben definiáltam a NetDriver-t:

```
[/Script/Engine.GameEngine]
+NetDriverDefinitions=(DefName="GameNetDriver",DriverClassName="OnlineSubsystemSteam.SteamNetDriver",DriverClassNameFallback="OnlineSubsystemUtils.IpNetDriver")
```

DefName: az egyedi neve a net driver definíciónak

```
[/Script/OnlineSubsystemSteam.SteamNetDriver]
NetConnectionClassName="OnlineSubsystemSteam.SteamNetConnection"
```

DriverClassName: ez az elsődleges driver osztály neve

DriverClassNameFallBack: ez annak az osztálynak a neve, ami a tartalék, ha az elsődleges driver nem megfelelően töltene be

Beállítottam, hogy az „Online Subsystem Steam” használja:

```
[OnlineSubsystem]
DefaultPlatformService=Steam
```

OnlineSubsystemSteam modul konfigurálás:

```
[OnlineSubsystemSteam]
bEnabled=true
SteamDevAppId=1929430
```

TD_Wave.build.cs -ben hozzáadtam a projekthez a steam modult:

```
DynamicallyLoadedModuleNames.Add("OnlineSubsystemSteam");
```

Engine-Inputs: létrehoztam a karakter mozgásért felelős bemeneti definíciókat, amelyek fogják a karakter 2D-ben mozgatni:

MoveForward:

- W lesz az előre ezért neki 1-es értéket adok
- S lesz a hátrafelé ezért ő az előrének a -1 szerese, azaz -1

MoveRight:

- D lesz a jobbra neki az értéke 1
- A lesz a balra ez pedig a jobbra -1 szerese

4. Játékos karakter létrehozása

Létrehoztam egy master_player Blueprint-et Paper Character-ből származtattam, ami tartalmazott egy Capsule Component-et, egy Sprite komponenst, egy Arrow Component-et, és egy Character Movement komponenst, mivel jelenleg egy játszható karakter van, és későbbiekben nem lesz funkcionalitásban eltérés a karakterek között (ha lesz több karakter), ezért nem származtatással fogom megcsinálni a karaktereket, hanem adat táblából fogom feltölteni a master_player karaktert.

4.1.Karakter alapértékek

4.2.Mozgás implementációja

A MoveFoward eseménynek az Axis értékét megvizsgálom, ha nagyobb mint 0 akkor az előre kell elmozdítani a karaktert, ha kisebb akkor hátrafelé. Az Add_Movement_Inupt függvénynek négy bemenetiértéke van nekem ebből kettő kell a World_Direction, ami megadja, hogy milyen irányba akarom a karaktert mozgatni, és a Scale_Vaule, ami megadja, hogy milyen mértékben akarom elmozdítani a karaktert, ez a MoveFoward Axis értéke lesz, World Direction-t ki lehet számolni a Get_Control_Rotation fügvény Z visszatérésiértékével a Get_Forward_Vector-t meghívni, és ez lesz a World Direction értéke.

A MoveRight eseménynek az Axis értékét megvizsgálom, ha nagyobb mint 0 akkor az jobbra kell elmozdítani a karaktert, ha kisebb akkor balra. Az Add_Movement_Inupt függvénynek négy bemenetiértéke van nekem ebből kettő kell a World_Direction, ami megadja, hogy milyen irányba akarom a karaktert mozgatni, és a Scale_Vaule, ami megadja, hogy milyen mértékben akarom elmozdítani a karaktert, ez a MoveRight Axis értéke lesz, World Direction-t ki lehet számolni a Get_Control_Rotation_fügvény Z visszatérésiértékével a Get_Right_Vector-t meghívni, és ez lesz a World Direction értéke.

4.3.State Machine

Sajnos Unreal Engine 5 nem támogatja a nem 3D animáció State Machine-t, ezért nekem kell létrehoznom egyet, ehhez létrehoztam egy state_enum-ot, amiben eltárolom a lehetséges állapotokat majd létrehozok egy anim_struct-ot, ami egy struktúra, és ebben lesz egy Flipbook referencia loop Boolean és a későbbiekre tekintettel egy Integer (notifyFrame), amivel majd

implementálni tudom az Animation Notification-t. Létrehoztam egy adattáblát az anim_struct alapján witch_anim_data névvel majd feltöltöttem adattal:

	Row Na	flipbook	loop	notifyFrame
1	idle	PaperFlipbook'/Game/player/witch/idle/witch_idle.witch_idle'	True	-1
2	attack	PaperFlipbook'/Game/player/witch/attack/attack.attack'	False	3
3	hit	PaperFlipbook'/Game/player/witch/hit/witch_hit.witch_hit'	False	-1
4	run	PaperFlipbook'/Game/player/witch/run/run.run'	True	-1

notifyFrame alapértelmezett értéke -1-et adtam meg, mert ha nem szeretném, hogy aktiválja az eseményt akkor vagy egy hatalmas számot kell megadni vagy egy negatívat.

Kibővítettem a mozgás logikát úgy, hogy mozgásnál állítsa a state nevű enum-ot run-ra majd meghívja az update_anim függvényt.

4.3.1. Update_anim

Az update_anim() függvény meghívja a Get_Data_Table_Row() függvényt, a két bementiértéke: witch_anim_data és a state, a visszakapot adatsort elemire bontóm: Flipbook, Loop, Notify Frame majd beállítom ezeket az értékeket Set Flipbook(Sprite,Flipbook), Set Looping(Sprite,Loop) és eljátszom a Flipbook-ot a Play(Sprite) függvénnyel.

4.3.2. Animáció értesítés kezelése

Az Update_anim függvényben megkapott Notify_Frame értéket kell megvizsgálnom, ha nagyobb mint -1 akkor az animáció tartalmaz animáció értesítést, ha a Branch(Notify_Frame) igazzal tér vissza akkor létrehozok egy timer-t , ami a Get_world_Delta_Seconds() visszatéri értékével megegyező időközönként hívja meg a anim_notify eseményt majd ezt a Timer_Handler változóban eltárolom. A anim_notify esemény-nek meg kell vizsgálnia hogy a Get_PlaybackPosition in Fames(Sprite) függvény nagyobb-e a Notify Frame értékénél, ha igen akkor beléphetünk a Do_Once node-ba, amit majd új Timer létrehozásánál visszaállítunk, majd meghívjuk a fire_event eseményt és a Clear and Invalidate Timer by Handle(Timer Handler)

4.3.3. Fire_event

A fire_event esemény egy switch on State_enum-ot hív meg, mert tudnom kell, hogy milyen animáció játszódott le és annak függvényében kell cselekedni.

4.4.Event AnyDamage Esemény

Az Event AnyDamage esemény akkor fog lefutni, ha a karakter 0-nál nagyobb sebzést kap. Amint a karakter sebzést kap át kell állítani a State-et hit-re azaz a karaktart éppen sebzik majd meg kell hívni a Update_anim() függvényt, de előtte meg kell vizsgálni, hogy éppen hit animációt játszunk, amit a Get_flipbook(Sprite) visszatérési értéke az nem egyezik meg a witch_hit-tel, ha igen akkor nem hívom meg a Update_anim() függvényt, majd a sebzés érékét csökkentem a karakter armor (páncél) érékével, ha a ez az érték kisebb lenne mint 1 akkor 1-re állítom be az érékét majd azt kivonom a current_hp változóból, ha az eredmény <=0 akkor veszített a játékos. Megállítom a játékot, majd meghívom a Game_over eseményt-et

4.5.Event BeginPlay esemény

Az Event BeginPlay az első esemény a konstruktor után. Én két Timer-t hozok itt létre az egyik generációért lesz felelős (regen esemény) a másik meg az animációk alaphelyzetbe állításáért (trail esemény)

4.5.1. Trail esemény

A karakter után elhalványuló körvonalak jelennek meg, hogy a mozgást szemléltesse.

Az effektet Niagara-val készítettem, ami az Unreal Engine-be épített VFX készítő rendszer.

Létrehoztam egy NiagaraEmitter-t player_VFX névvel, ami a Fontain Emitter sablont használja. Töröltem a Spawn_Rate, Shape_Location, Add_Velocity, Gravity_Force, Drag, Solve_Forces_and _Velocity komponenseket.

Hozzáadtam a Spawn_Burst_Instantaneous, Inital_mesh Orientation, Sub_UVAnimation komponenseket.

Módosítottam a Loop_Behavior-t a Emitter_State-ben „Once”-ra. A Sub_UVAnimation-ban módosítottam az End_Frame-et 7-ra, mert 8 frame hosszú az animáció és 0. frame-nél kezdődik, majd beállítottam a Sprite_Renderer-ben a Material-t a player_material-ra.

5. Material

5.1.player_material

Létrehoztam egy material-t, módosítottam a Blend_mode-ot „Translucent”-re, hozzáadtam egy „TextureSampleParameterSubUV” elneveztem Flipbook-nak, majd beállítottam a flipbook értékét „B_witch_run(texture)”