Szakdolgozat

Godot Engine

A Godot Engine egy nyílt forrás kódú játék motor amit a közössége fejleszt már (TODO Dátum vagy több mint x éve) óta. A már más motorokban megszokott "Entity component" rendszer helyett egy "node" alapú rendszeren használ, ami hasonló a Blender (szintén nyílt forrású) program rendszeréhez. 2 illetve 3D -s játék fejlesztésre is használható, alábbiakban leginkább a 2D fejlesztésről lesz szó.

Node rendszer

"Minden node". Az Godot engine -en belül készült objektumok nagy része node -ok által vannak reprezentálva, van ez alol néhány kivétel például a "resource" -ok pl.: képek, videó, animációs fájlok, szkript fájlok stb. A lényeg hogy egy Godot -ben készült játék egy "Scene Tree" -re épül, lényegében egy fára és minden node a fának a része hívatnánk őket akár gráf pontoknak is, mert lényegében azok. Tehát minden node -nak van egy "parent" -je, és egy vagy több gyereke. Tudunk készíteni rész fákat is (.tscn/.scn) amiket aztán többször is használhatunk a projektünkben. Így például létre tudunk hozni komponenseket is hasonlóen egy "Entity Component" rendszerhez. Viszont itt fontos hogy egy node a fában hol helyezkedik el, pl.: ha van két TextureRect node -unk aminek két különbözű texturája van, akkor amelyik később van a fában az lesz később, vagyis a 2. el fogja takarni az 1. mert az került renderelésre később ezt a működést a "z-index" vátoztatásával lehet befolyásolni, 3D esetén ez nem igaz mivel ott máshogy működik a renderer- és viszont ebben az esetben is először az 1. node fog lefutni.

Entity Component rendszer

GDScript

A Godot Engine -be épített szkript nyelv, szintaxis -ban hasonlít a python -ra viszont azon kívül hogy van egy "Variant" típus ami alapértelmezett minden változóra, ez egy típusos nyelv, viszont csal opcionálisan az, vagyis a typus meghatározások elhagyhatók.
(TODO kép)
Vagyis a "Variant" esetében az engine próbálja meg "kitalálni" compile time hogy mi lehet a változonk típusa, ha viszont megadjuk a típusát akkor az engine szól hogyha esetleg mégsem az a típusa amit megadtunk a változónak. Alapvetően mindent meg lehet csinálni GDScript -ben, persze amit az engine enged. Ez egy Object oriented programozási nyelv, habár első látásra nem biztos hogy annak néz ki, de ez a legtöbb game engine -en belül készült szkriptre is igaz, mert általában ugye nincs semmilyen "main" függvény. A legfontosabb metódusok amiket felül lehet írni azok a _ready, _process és az _input függvények, a _ready akkor hívódik meg amikor a node beérkezik a "Scene Tree" -be és lefutott az init vagy _init függvénye (fontos hogy az init -et nem tudjuk felülírni és nem is látszódik, viszont az _init -et igen), más szóval ha a node készen áll a használatra, a _process a _ready után kezd el futni, de nem a _ready futattja, a _process

Előnyök, Hátrányok

GDNative

A GDNative lényegében olyan mintha a forráskódot módosítanánk picit limitáltabb módon, C vagy C++ nyelven.
Érdemes megjegyezni hogy a Godot, nem CMake -et használ hanem Scons -t ami egy píthon ban írt fordító, vagyis a fordítással kapcsolatos példák és egyebek azok Scons -ban vannak megcsinálva, természetesen lehet CMake -el is fordítani van is rá már azóta példa, de nem "official" támogatott. Kódolás szempontjából ez nem változtat semmin. A compile eredménye egy ".gdnative" fálj és egy ".dll" fájl amit ha azután beteszünk a godot projectünk egy mappájába akkor az engine autómatikusan felismeri (hot-reload) és futatja a kódunkat, tehát "bövítjük" az engine -t. A forráskód módosítás után ez a leg szabadabb módja a Godot -ban való kódolásnak, mert így sokkal több mindent elérünk, és sokkal hatékonyabb az így megírt kód, persze ezt a módszert inkább bövítmények használják, vagy olyan esetekben amikor valami nagyon erőforrás igényes funkciót készítünk.

CSharp Godot -ban

Jelenleg a .Net 8 -as verzió a követelmény. Általában véve nagyobb projektekhez ajánlott.

Más nyelvek Godot -ban

A 4 alapból támogatott nyelv mellett, a Godot Támogat más programozási nyelveket is nem "official" modulokon, addonokon keresztül, pl.: Python, Java. (TODO Összenyüjteni egy listát vagy a godot docs ban lévőket belinkelni)

Programozási nyelv kompatibilitás

A nyelveket lehet keverni, úgy hogy az egyik node szkriptjét az egyik nyelven másikat a másikban, GDNative esetében az ez által készített node -okat lehet tovább bővíteni.

Miért Godot?

MIT vagyis nyílt forrás kódú. (TODO ennél azért több érv kell)

Miért 2D?

Shader -ek

Particle -ök

"Game programming patterns" / "Design patterns"

Singleton

Beállítások, Mentés menedzsment, Globális változók, tárgyak betöltése

State machine

Játékos illetve ellenfélnél hogy van használva

Út keresés

Navigation server, AStar (corridor funnel, edge centered)

A Godor támogat obstacle avoidence -et a 4.x es verziók óta, viszont ezen projekt esetében ilyenre nem volt szükség mivel az ellenfelek egymással, vagy a játékossal nem érintkeznek, vagy legalábbis nem olyan szinten, arra egy külön rendszer van implementálva (hurtbox/hitbox), egy felül nézetes játékhoz használhatóbb lenne.

Játék mentés / Betöltés

A játék mentését először egy egyszerű rendszerrel oldottam meg amelyre van is egy példa a Godot dokumentációs oldalán, a rendszer lényege hogy meghatározunk egy "persistent" nevű node csoportot aminek annyi a lényege hogy így a "Scene Tree" -ből bárhonnan lekérhetjük az összes ebbe a csoportba tartozó node -ot, minden csoportban lévő node -ra definiálunk egy mentés függvényt, ennek a függvénynek ami minimum amit vissza kell adnia az a filename és a node parent -je, vagyis hogy milyen node -ot mentünk, és hol van ez a node, ezután készítünk egy singletont, az autoload projekt beállítás segítségével, ez az autoload egy sima node -ból származó szkript lesz amiben készítünk egy mentés illetve betöltés függvényt, a mentés függvény tartalma csupán annyi hogy lekérjük a "persistent" node csoportban lévő összes node -ot, meghívjuk a mentés függvényüket amik visszaadnak egy "dictionary" adat típust, ezeket összefüzzük aztán Json -ra konvertáljük.

A betöltést ebben az esetben úgy kell implementálni hogy le kérjük ebben az esetben is a "persistent" csoportban levő node -okat, betöljük a mentet Json fájlunkat, és ezután kitöröljük az összes node -ot a "persistent" csoportban utánna a fájl visszakonvertáljuk egy "dictionary" -re az adatok alapján betöljük a node -okat és be inicializáljuk őket a mentett adatokkal. Azért fontos hogy kitöröljük a régi node -okat, egyrészt hogy ne keljen a mentés és a csoportban lévő node -ok sorrendjével bajlódni, másrészt meg így biztos hogy helyesen lesznek a node -ok paraméterei hiszen lehet hogy betöltéskor már nem alapállapotban van a node.

Később ezt a rendszert úgy módosítottam, hogy nem Json formátumban vannak mentve az adatok hanem egy "custom resource" objektumban ami így lehetővé tette hogy egyszerűen mentsem a karakter "INVENTORY"(szép magyar szó ez) tartalmát, tehát mostmár egy .tres fájl formátumban van ami egy text formátum, de ezt bármikor meg lehet változtatni binaris (.res) formátumra. Megjegyezném hogy egyébként sok játék van ami nem figyel a mentés fájlok védelmére, vagyis könnyű a mentés fájl módosításával előnyökhöz jutni.

Tábor rendszer

Crafting system

Platformer

Ellenfelek

Fő ellenségek