**Szegedi Tudományegyetem**

**Informatikai Intézet**

**SZAKDOLGOZAT**

**Galgóczki Norbert**

**2021**

**Szegedi Tudományegyetem**

**Informatikai Intézet**

**Három dimenziós labirintus játék megvalósítása Unityben**

**SZAKDOLGOZAT**

|  |  |
| --- | --- |
| Készítetté:  **Galgóczki Norbert**  programtervező informatikus  hallgató | Témavezető:  **Jász Judit** |

**Feladatkiírás**

* A játék indításakor az automatikusan generál egy labirintust
* A generált labirintusból egy 3D színteret épít a generáló algoritmus segítségével
* A generált színtéren a játékos egy avatár irányításával navigál majd és járja be a labirintust
* Pálya generáló, amely saját generáló algoritmust használ
* A pálya generálónak generálnia kell:
  + magát a labirintust
  + kijáratot
  + nagyobb termeket/szobákat
* Labirintus automatikus berendezése:
  + falra rögzített fáklya
  + asztal
  + szék
  + szekrény
  + könyvek
  + mennyezetről függő lámpa
  + gyertya az asztalra random generálva
  + ajtó
  + kinyitható ládák
  + ládákban egy felhasználható csont
* Ellenfelek implementálása, célja a játékos elkapása:
  + ellenfél kutya
  + a kutyát fel lehet tartani a csontot használva
* A játékos célja:
  + kijusson a labirintusból
  + elmeneküljön a kutyák elől

# **Tartalmi összefoglaló**

* A téma megnevezése:
  + Három dimenziós labirintus játék megvalósítása Unityben
* A megadott feladat megfogalmazása:
  + Unityben egy olyan Szintér felépítése, melyben egy karaktert lehet irányítani a háromdimenziós labirintusban, melyet saját generáló algoritmussal van megvalósítva. A labirintusban kell lennie kijáratnak és nagyobb szobáknak. A labirintusban kutyák elkerülése közben kell kijutni a kutyák mellet csontok felhasználása segítségével lehet elhaladni, melyeket a labirintusban elhelyezett ládákból lehet megszerezni.
* A megoldási mód:
  + Elkészítettem a generáló algoritmust és a Unity motor segítségével elkészítettem a labirintus, majd az irányitó scriptekkel játszhatóvá tettem a játékot.
* Alkalmazott eszközök, módszerek:
  + A játék készítésé során a Unity által kapott eszközöket használtam a játék elkészítéséhez és Blendert használtam a modellek és animációk elkészítéséhez, amit importáltam a Unitybe
* Elért eredmények:
  + Játszató verzió elkészült és fel van téve publikusan githubra.
* Kulcsszavak:
  + Unity, Blender, Labirintus, Játék, Generáló algoritmus

**Tartalomjegyzék**

[**Feladatkiírás** 2](#_Toc71830239)

[**Tartalmi összefoglaló** 3](#_Toc71830240)

[**Bevezetés** 6](#_Toc71830241)

[**1.** **Unity** 7](#_Toc71830242)

[**1.1.** **Játékmotorok** 7](#_Toc71830243)

[**2.** **Blender** 8](#_Toc71830244)

[**2.1.** **Modellező programok** 8](#_Toc71830245)

[**2.2.** **Blender alapok** 8](#_Toc71830246)

[**3.** **Fényforrások** 12](#_Toc71830247)

[**3.1.** **Fényforrások alapjainak ismertetése** 12](#_Toc71830248)

[**3.2.** **Point light** 12](#_Toc71830249)

[**3.3.** **Spot light** 13](#_Toc71830250)

[**3.4.** **Directional light** 13](#_Toc71830251)

[**3.5.** **Area light** 14](#_Toc71830252)

[**4.** **Labirintusok** 15](#_Toc71830253)

[**5.** **Labirintus generáló algoritmusok** 16](#_Toc71830254)

[**6.** **Saját labirintus generáló algoritmus** 16](#_Toc71830255)

[**7.** **Modellek készítésé** 17](#_Toc71830256)

[**7.1.** **Fáklya, gyertya és csillár** 17](#_Toc71830257)

[**7.2.** **Szekrény, könyv, asztal és szék** 18](#_Toc71830258)

[**7.3.** **Kutya** 19](#_Toc71830259)

[**8.** **Scriptek** 20](#_Toc71830260)

[**8.1.** **Button Manager script** 20](#_Toc71830261)

[**8.2.** **Player script** 20](#_Toc71830262)

[**8.3.** **Torch script** 21](#_Toc71830263)

[**8.4.** **Dog script** 21](#_Toc71830264)

[**8.5.** **Ajtó script** 22](#_Toc71830265)

[**8.6.** **Láda script** 22](#_Toc71830266)

[**8.7.** **Mazegenerater script** 23](#_Toc71830267)

[**9.** **Animációk** 24](#_Toc71830268)

[**9.1.** **Játékos animációja** 26](#_Toc71830269)

[**9.2.** **Kutya animációja** 26](#_Toc71830270)

[**9.3.** **Láda animációja** 27](#_Toc71830271)

[**9.4.** **Ajtó animációja** 27](#_Toc71830272)

[**10.** **Unity prefabok** 28](#_Toc71830273)

[**10.1.** **Csont prefab** 29](#_Toc71830274)

[**10.2.** **Láda prefab** 29](#_Toc71830275)

[**10.3.** **Kutya prefab** 30](#_Toc71830276)

[**10.4.** **Fáklya prefab** 30](#_Toc71830277)

[**10.5.** **Utolsó szoba prefab** 31](#_Toc71830278)

[**Nyilatkozat** 33](#_Toc71830279)

# **Bevezetés**

Szakdolgozatom elkészítésénél az volt a célom, hogy megtanuljam és megismerjem egy 3D játékprogram elkészítésének folyamatát. A programozási lehetőségeket, annak menetét és a programok működését. Ezen belül a választott programmotor „Unity” milyen lehetőségeket biztosít és persze nem utolsósorban az, hogy képes legyek készíteni egy használható 3D játékot. Gyermekkorom óta érdekeltek a számítógépes játékok és most a szakdolgozat egy remek alkalomnak ígérkezett, hogy készítsek egy számítógépes játékprogramot. Egy olyan játékprogramot próbáltam megalkotni, amely a felhasználója részére kellően érdekes lehet és élvezhető játékélményt nyújt. A játékprogram megírása során próbáltam lehetőséget biztosítani arra, hogy az elkészített pálya később egy nagyobb, komplexebb játék részévé is válhasson. Úgy építettem fel a programot, hogy könnyű legyen továbbfejleszteni, vagy esetleg használni az elemeit.

A játékprogram elkészítésénél először megnéztem, hogy milyen labirintus generáló algoritmusok vannak és azok közül melyek az ismertek. Próbáltam olyan algoritmust keresni, amit megpróbálhatok továbbfejleszteni és olyan labirintusokat tudok generálni velük, ahol a labirintus bármely pontjából elérhetem a másikat.

Miután az algoritmust kitaláltam felépítettem rá a pálya generáló algoritmust. Az algoritmus elkészítésének szempontjainál ügyeltem arra, hogy a szakdolgozati feladatban megnevezett elemeket tartalmazza és töltse be a scene térre. Ezek után adtam hozzá a játékost, aki be tudta járni a scenetérben elhelyezkedő labirintust.

Ezek után elkészítettem a játékhoz a kezdő felületet, ahonnan be lehet lépni a labirintusba és ha a játékos kijutott a labirintusból akkor ide tér vissza. Ezen a kezdőlapon kapott helyet a beállítások felület, ahol be lehet állitani a játék hangerősségét és az egér érzékenységét.

A kezdőlapon lehet elolvasni a játék történetét, melyből megtudhatjuk, hogy a játékban kik vagyunk és milyen cél elérésért kell végig haladnunk a labirintuson.

# **Unity**

# **Játékmotorok**

A játékmotorok azért készülnek, hogy segítsék a játékfejlesztő munkáját a játékprogramok készítésénél. A motorok már tartalmaznak, olyan alapfunkciókat, melyek mindig felmerülnek egy játék készítésé során és azok egy részét beavatkozás nélkül megoldják.

Ilyen alapfunkciók: a játék területének létrehozása, az objektumok elhelyezése és azok kezelése, kezeli továbbá a különböző hangfájlokat és a hangokat, a kamerát és rendezi a textúrákat, valamint kezelik a különböző kiterjesztésű fájlokat, mint például a Blendert és igy tovább. A játékmotorok gondoskodnak arról, hogy az alap alkotóelemeket összekössék, melyből a fejlesztők fel tudják építeni a saját világukat. Rengeteg különböző játékmotor jött létre az évek során. Vannak vállalatok, akik maguknak fejlesztett ki egy egyfajta modellező programot, saját játékmotort, melynek elég csak egyfajta specifikus fájlt beolvasnia ezzel csökkentve a motor méretét és növelve az olvasási sebességet. Azonban léteznek olyan játékmotorok, melyek kifejezetten azért készülnek, hogy a játék készítőknek minél szélesebb körét ki tudják elégíteni. Ezek a motorok a lehető legtöbb fájlt képesek kezelni és a lehető legtöbb dologban tudja segíteni a fejlesztőket a játékuk készítésében.

* 1. **Unity alapok**

Unity az egyike a legismertebb játék motoroknak az Unreal motor mellet. Minden évben rengeteg játékot készítenek a Unity játékmotor használatával. (például: 2020 évben jelent meg az egyik kedvenc játékom is, ami ezen a motoron alapszik a Genshin Impact)

A Unity motort is folyamatosan fejlesztik és évről-évre jönnek ki a frissítések és újabb verziók. A Unity igyekszik a lehető legtöbb módon segíteni a fejlesztőket, ezért támogatja a nagyobb modellező programok kiterjesztéseit is, mint például a Blender, a Modo és az általános fbx. kiterjesztés. A Unity a fény kezelésnél is rengeteg grafikai beállítást és lehetőséget biztosít a játéktervezők részére.

# **Blender**

# **Modellező programok**

Általában a modellek alatt háromdimenziós modelleket értünk és ezek megalkotásához használatosak a modellező programok. A modellek elkészítéséhez négy módszer létezik, a **Poligon modellezés**, amely a háromdimenziós térben elhelyezett pontok által alkotott poligonhálót hoz létre. A mai modellek legnagyobb része manapság textúrázott poligonmodell, mivel ezek rugalmasan alakíthatóak, és mert a számítógépek könnyen és gyorsan rendezik őket. Még azzal együtt is, hogy a poligonok síkbeli alakzatok, így sok poligon felhasználásával is csak megközelíteni tudjuk a görbe felületek ívét.

A második a **NURBS** ami az angol „Non-Uniform Rational Basis Spline”-nek a rövidítése ami tükör fordításban „nem uniform, racionális B-spline görbékkel definiált felület” modellezés – a NURBS felületeket súlyozott kontrollpontok által befolyásolt szplájn függvények görbéi definiálják. A görbék követik, de nem feltétlenül érintik a pontokat. A NURBS-felületek nem csak közelítik a görbületet kisméretű lapos felületekkel, hanem tényleg simák, így különösen alkalmasak organikus modellek készítésére. A Maya az egyik legismertebb kereskedelmi szoftver, amely natívan támogatja NURBS alkalmazását.

A harmadik a **Splines & Patches** modellezés – a NURBS-hoz hasonlóan ez is görbék segítségével ábrázol. A használat egyszerűségét és a rugalmasságát tekintve a poligon modellezés és a NURBS-modellezés közé esik.

Végül a **Primitív modellezés** – Ez a modellezési módszer geometriai primitíveket vesz alapnak, mint például gömbök, hengerek, kúpok vagy síkok, ezekből épít fel komplexebb alakzatokat. Az előnye az, hogy gyors és könnyű használni. A méretek abszolút pontosak, mivel a formák matematikailag definiáltak, ezen kívül a leíró nyelve is egyszerű. Ez a módszer jól alkalmazható technikai jellegű problémákra, és kevésbé jó az organikus dolgok modellezésére.

Néhány 3D alkalmazás direkt módon tud renderelni primitívekből, ilyen például a **POV-ray**, más programok pedig csak szerkesztési eszközként alkalmazza, az objektumot a későbbi műveletekhez poligonhálóvá konvertálja.

# **Blender alapok**

Az egyik legnépszerűbb modellező program a Blender, melyre az egyetemi tanulmányom során részleges ismereteket szereztem. Az egyetemen az alap irányitásokat vettük át, de a modellező programok manapság már számos fejlesztésen esett át és sok mindenre használható. A Blendert napjainkban már nem csak modellezésre használják, hanem rengeteg esetben találkozhatunk a program segítségével elkészített animációkkal. A Bender-ben az objektumokat hierarchikus rendszerben látjuk, ahol tudjuk menedzserelni a különböző objektumainkat.  
A Blender-ben amikor új fájlt hozunk létre mindig három komponenst fogunk látni:

1. a kiinduló alakzatunkat egy egyszerű kockát
2. egy fényforrást, amely a kezdő alakzat jobb felső sarkánál helyezkedik el és egy fehér színű point light fényt, ami abból a pontból kifele sugározza a fényt.
3. egy kamerát, amely az alakzat felé néz.

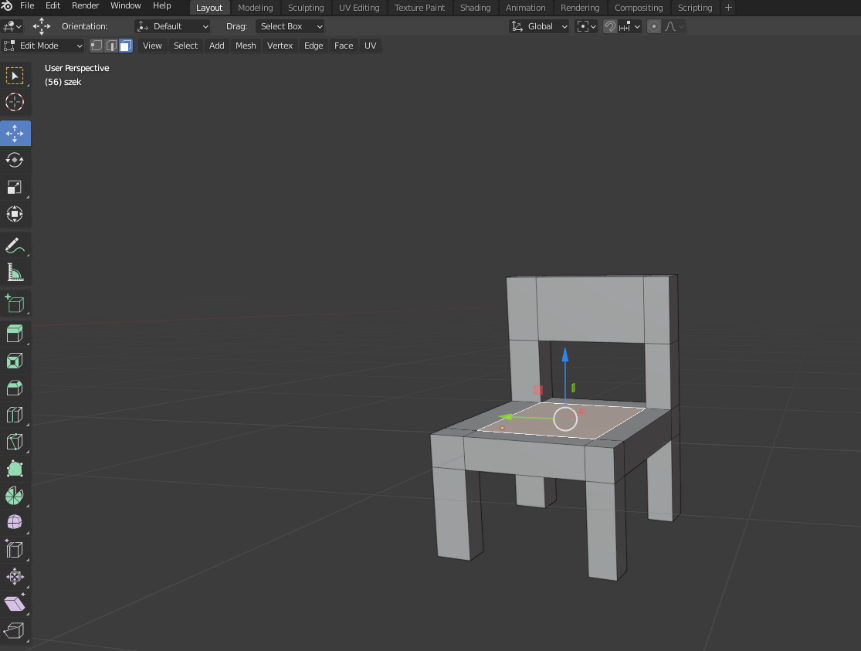
Az alakzat formázása oly módon történik, hogy az új alakzat pontokból áll és ezeket a pontokat szélek kötik össze. A szélekből lesznek az oldalak ezeket hívjuk Vertex-ek, Edge-ek és Face-eknek, melyekre később hivatkozni fogunk.

A Blender-ben több nézet is elérhető, melyek a különböző célok elérésére szolgának és segítik az alkotót. Főbb ablakok közé tartozik az Object mode, melyben az objektumainkat tudjuk menedzselni. Ebben a mode-ban az alábbi főbb funkciókat tudjuk használni:

1. a forgatás másnéven Rotate
2. a mozgatás az-az Move
3. az átméretezés az-az a Scale, ahol a kiméréseket akár vonalzóval is tudjuk segíteni.

Ezek mellet hozzá tudunk adni új objektumot, illetve lehet jegyzetet szerkeszteni, melyet a háromdimenziós térbe tudunk elhelyezni.

Ez után következik az Editor mode ahol az object mode-ban kiválasztott alakzatot tudjuk átalakítani. Az Editor mode-ban is megtalálhatók az előzőekben taglalt eszközök, melyek kiegészülnek még pár újabb funkcióval, melyek az 1-es ábra bal szélén találhatók.



1. ábra 2.2 fejezet: Blender kezelőfelülete.

A leggyakrabban használt új funkciók a következők:

1. Extrude Region ahol attól függően hogy Vertexet vagy Edget vagy Facet oldalt jelöltünk ki, azt fogja kiemelni.
2. az Insert Faces ahol a kijelölt Facek alapján egy új facet hoz létre
3. a Bevel, ahol a kijelölések alapján a kurzor irányításával kivág egy szeletet az alakzatból
4. a Loop cut ami az alakzat körül egy hurkot képez és ott elvágja az alakzatokat és az Edgek mentén amit érintett új Vortexeket hoz létre és köti őket össze.

Ezeken kívül van még emellett hét fő eszköz, amit lehet használni de ez a négy a leggyakrabban használt eszköz.

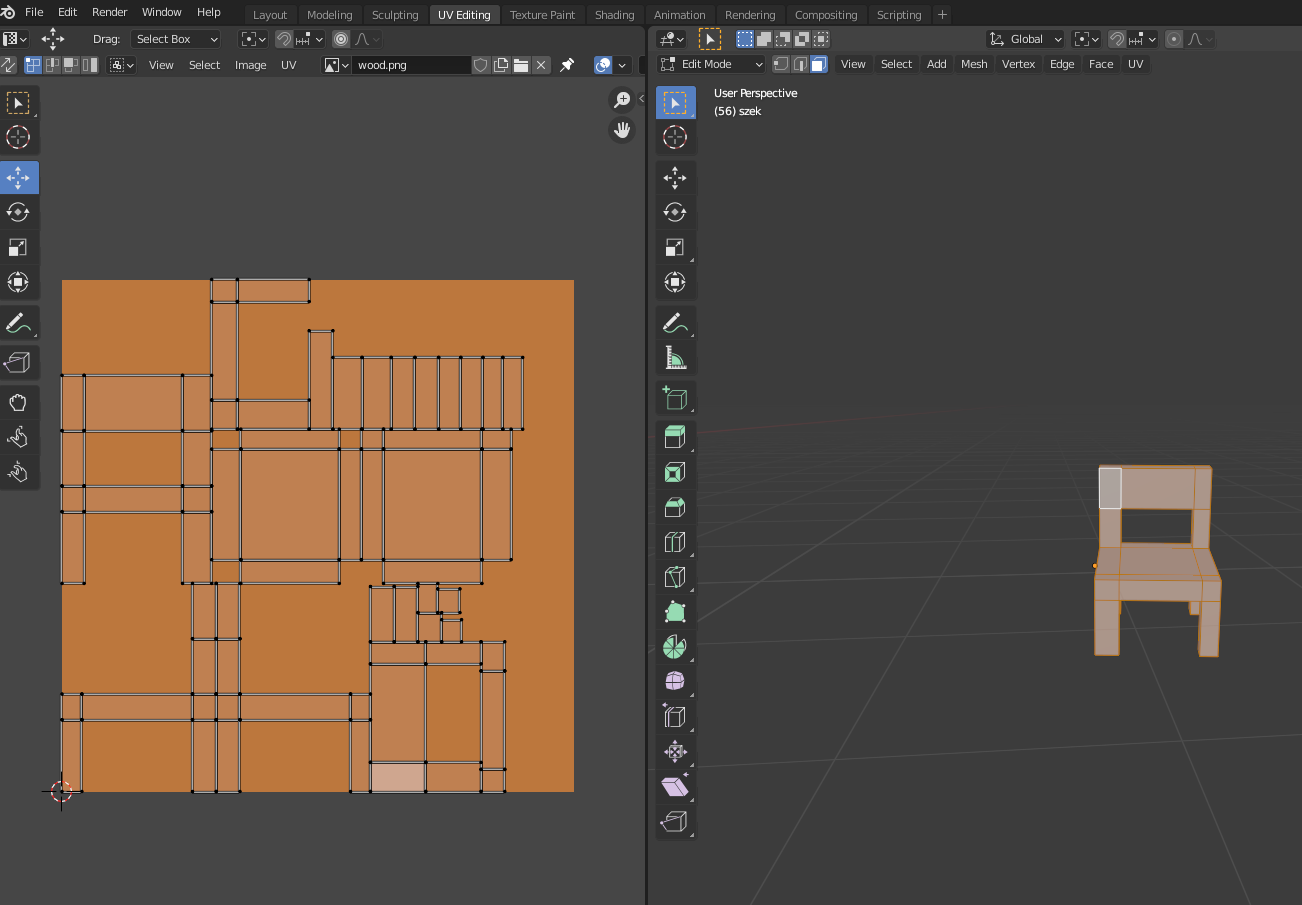
A rengeteg mode közül, mellyel a Blender rendelkezik még hármat kívánok kiemelni, melyeket gyakran használtam a dolgozatom elkészítésé során.

Ezek a következők:

* az UV Editor
* a Textura painter
* Animation mode

Ezekkel a funkciós lehetőségekkel tudtam az elkészült alakzatok textúráját az-az kinézetét véglegesíteni az UV Editorban. A mód alkalmazásával be tudjuk állitani azt, hogy az alakzatunk egyes Face-nek a textúrája milyen legyen és hol helyezkedjen el az UV térképen, A mozgatás, forgatás és átméretezés eszközünk az UV Editor-ban is rendelkezésünkre áll, melynek segítségével tudjuk a térképen elhelyezni a Facek Vertexeit, hogy majd megfelelő képe legyen és ne legyen elcsúszva. A Facek-nek az UV térképen történő elhelyezésére három lehetőségünk van:

* mi kézzel elhelyezzük a Face-t
* szegéseket állíthatunk be, hogy a számítógép miképp hajtsa szét az alakzatunkat, hogy az kétdimenzióson elférjen, mint ahogyan a kisgyerekek a kockát hajtogatják ugyan így mi is be tudjuk jelölni ezt az alakzatunkon, ezt a lehetőséget a 2-es ábra mutatja be
* vagy beállíthatjuk, hogy úgy vetítse le kétdimenziósra ahogy azt mi látjuk a képszerkesztőben



. ábra, 2.2 fejezet: feldarabolt szék modell

Mind a három megoldásnak megvan, hogy mikor a leghasznosabb.

Amikor készen vagyunk az UV térkép elkészítésével a Texture Painter ablakba be tudjuk importálni azt a képet, amelyre terveztük az UV térképet. Vagy akár ebben a fülben mi magunk is elkészíthetjük a texturához a saját képünket. A kép szerkesztéséhez a Blender is kínál alap funkciókkal rendelkező eszközöket, amely elégséges a Textura elkészítéséhez, de az mégsem egy professzionális rajzfelület.

Az alakzatunkhoz hozzárendelhetünk csontokat, melyeknél a program automatikusan kiszámolja, hogy az alakzatunk mely szakaszai csatlakoznak a csonthoz és milyen erősen.

Ezt a Weight Paintel lehet alakítani úgy, hogy ha azt látjuk, hogy az automatikusan kiszámolt súlyozás nem megfelelő az alakzatunkhoz, azt megváltoztathatjuk. Amennyiben elégedettek vagyunk a csontok elhelyezkedésével és súlyozásával, akkor a Pose mode-ban rögzíthetjük.

A csontok alapján tudjuk az alakzatunk beállását változtatni és ennek segítségével meg tudjuk animálni és a kamerával felvehetjük a meganimált jelenetet. Az animációt akár ki is exportálhatjuk FBX. formátumba vagy esetleg importálhatunk már meglévő animációkat, képeket, mivel a Blender rengeteg formátumot támogat.

# **Fényforrások**

# **Fényforrások alapjainak ismertetése**

A játékoknak és a modellezésnek egyik nagyon fontos eleme a fény és a fénnyel való játék. A fénynek és a fényforrásoknak minden esetben van erőssége és színe. A legalapvetőbb fény, ami nem sorolható a fényforrások közé, de a modellezésnél elég fontos, az az Ambient fény, ami egy környezeti fény. Egy olyan, fény amely mindenhol jelen van és nem származik semmilyen forrásból. Ennek a fajta fénynek nincs sem iránya, sem távolsága.

A grafikában négy fényforrás fajtát ismerünk:

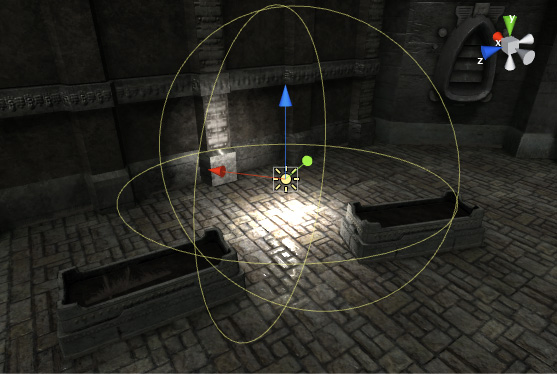
* a Point lightot
* a Spotlight
* a Directional light
* a Area light

Ezeknek már van különlegeségük mint például van irányuk, van forrásuk és igy tovább.

A Unity és a Blenderben is ezeket ismertjük és használjuk.

# **Point light**

A Point light egy olyan fényforrás, amely egy pontból világít a térbe minden irányba és egyenlő erősséggel, mely a 3. ábrán látható.

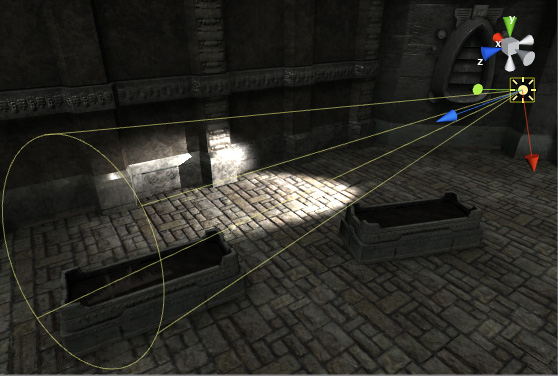


3. ábra, 3.2 fejezet: Unity Point light

A fény ahogy távolodik a forrásától veszít az intenzitásából. A fényintenzitás fordítottan arányos a forrástól mért távolság négyzetével. Ezt „inverz négyzet törvénynek” nevezik, és hasonló ahhoz, ahogy a fény a való világban viselkedik. A Point light-ot általában lámpáknál, tüzeknél és robbanásoknál szokták használni, ahol a fény minden irányba terjed.

# **Spot light**

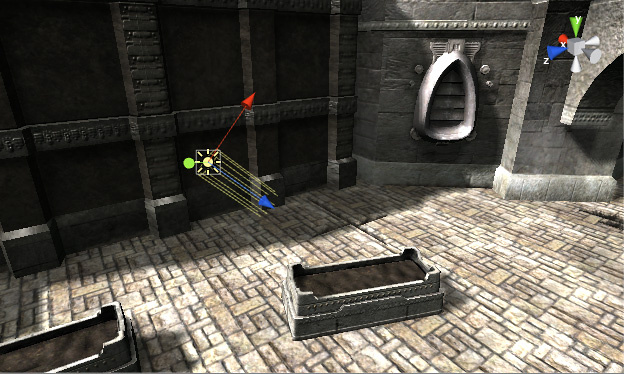
A Spot light nagyban hasonlít a Point light-hoz, rendelkezik minden Point light jellemzővel. A fény erőségével, a fény színével, annak térbeli helyzetével, azzal, hogy milyen messzire ér el a fény, ezeken felül a Spot light rendelkezik még egy szög jellemzővel is. A szög jellemző azt jelenteni, hogy a Spot light-ból mekkora szélességben világít a fény, mint a valóságban is egy reflektornál vagy egy zseblámpánál. A fény nem minden irányba halad hanem csak egy bizonyos szögben. A spot light fénynek a szélénél csökken a fényerőség. Minél nagyobb a fényforrás szöge, annál jobban mosódik el a fény a Spot light kúpjának a szélén, ez az angol terminológiában „penumbra” -nak nevezik. A Spot light fényforrást általában reflektoroknál, zseblámpáknál illetve kamerák fényénél szoktak alkalmazni. A 4. ábrán a spot light fény megjelenítése látható.



. ábra, 3.3 fejezet: spot light fény

# **Directional light**

A Directional light egy alapvető fényforrás. Amikor létrehozunk egy új Blender, Unity 5 vagy magasabb versionál projektet abban szerepel általában egy Directional light objektum. Ezt a fajta fényforrást Blender-ben Sun light-ként lehet megtalálni. A directional light, olyan helyzetekben hasznos amikor egy nagyon távoli fényforrást szeretnénk elkészíteni, mint a Nap, amely rendkívüli távolságban van. Ennek a fényforrásnak nincs semmi azonosítható forráspontja, ezért magát az objektumot akárhol elhelyezhetjük a színterünkön. Mivel itt a fényforrás pozíciója nem számít igy bármely objektumtól vett távolsága mindig egyenlő lesz, igy a távolság a fényforrástól és az objektumtól nincs meghatározva, ezért a fény ereje sem változik, igy állandó az erőssége. A Unity-ben ahogy forgatjuk ezt a fényforrást, napszakokat illusztrálhatunk, vagyis, ha felfele fordítjuk a fényforrást akkor besötétedik, mintha éjszaka lenne. A Directional light fényforrását az 5. ábrán fogom bemutatni.



. ábra, 3.4 fejezet: Unity Directional light

# **Area light**

Az Area light-ot általában a háromdimenziós térben lévő téglalapként szokták ábrázolni. Az Area light ennek a téglalapnak az egyik oldaláról bocsátja ki a fényét minden irányba akár csak a Point light. Ennek a fényforrásnak, mint a többi fényforrásnak a fénye a távolság négyzetével fordítottan arányosan veszít az fény erősségéből, kivétel a Directional light. Az Area lightra a 6.-os ábra mutat példát.

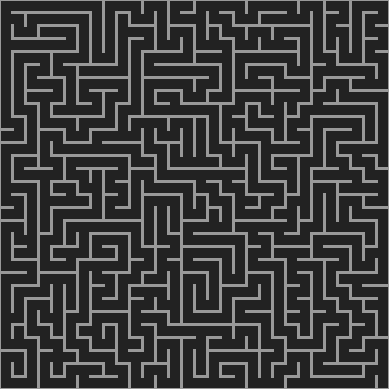
A képen beltéri, világos, sötét, lézer látható

Automatikusan generált leírás

. ábra 3.5 fejezet: Unity Area light

# **Labirintusok**

A labirintusok, olyan építmények vagy objektumok, melyek általában tartalmaznak egy vagy több bejáratot és egy vagy több célt, ami lehet akár a kijárat a labirintusból vagy valamilyen terület, ahova el lehet jutni. A labirintusok általában fejtöröknek készülnek, ahol a játékos megpróbál a bejárattól eljutni a célig. A labirintusok manapság elég népszerű játékok és rengeteg készül belőlük. Van, ahol egyszerűen csak annyi a cél hogy eljussunk a kijelölt területre, de vannak játékok ahol a kijutáson kívül célunk, hogy a lehető legrövidebb úton jussunk el a kijáratig. A bonyolultabb labirintus játékokban már nem csak a kijutás a cél, hanem el kell kerüljük a veszélyes objektumokat miközben a labirintus egyes szakaszai változnak. Így az egyszerű sémákkal már nem lehet könnyen kijátszani a labirintust, mint például azzal, hogy mindig jobbra fordulunk. A Labirintusok rengeteg stílusban léteznek. A 7.-as ábrán egy egyszerű hétköznapi labirintusnak az ábráját fogom bemutatni.



. ábra, 4. fejezet: Egyszerű labirintus

# **Labirintus generáló algoritmusok**

A labirintus generáláshoz négy különböző alapmódszert ismerünk.

Ezek a **Gráf elméleti** módszerek, a **Rekurzív osztás** módszer, az **Egyszerű algoritmusok** és a **Sejtszerű automata algoritmusok**. Ezeknek a módszereknek a nevei nagyon sokat elárulnak a felépítéseikről és működésükről.

**A rekurzív osztás** módszernél a labirintusban úgy határozzuk meg a falakat helyzetét, hogy a teret random felosztjuk egy fallal. A falon van egy átjáró így az újonnan elválasztott két területet újból feloszthatjuk addig amig végül már nem tudjuk tovább osztani.

**A Sejtszerű automaták** az élet játékából ismerhetőek, ahol egy cella attól függ, hogy él vagy nem, hogy a körülötte lévő cellák élnek-e és ezen a szabály alapján alakul ki.

**Az Egyszerű algoritmusok** mindig a labirintus csak egy szakaszát tárolják és úgy alakítják ki a labirintust.

A **Gráf elméleti** módszerek, ahol gráf elméleti módszereket használva generálják a labirintust. Mivel gráfokon alapul, ezért sokkal könnyebb átültetni ezeket az algoritmusokat, hogy ne csak táblázatokon működjenek, hanem sokkal absztraktabb labirintusokat is készíthessenek.

# **Saját labirintus generáló algoritmus**

A saját algoritmusom kigondolásánál a Gráf elméleti módszereket alkalmaztam. Azon belül is a mélységi keresés volt az, amit alapként használtam. A labirintusban, amit generálnia kell az algoritmusnak abban lennie kell szobáknak. Amennyiben a labirintust generálnám le először és abba helyezném be a szóbát, akkor olyan útvonal alakulna ki, amiket a játékos nem képes elérni, mert el van zárva. „zsákutca” Minél több szobát generálunk a labirintusba akkor annál nagyobb valószínűséggel kijátszhatatlanná válna a labirintusunk. Ezért először az elkészített labirintusba belegeneráljuk a szobákat, majd erre generáljuk rá a labirintus többi részét. A labirintus generálása a mélységi keresésen alapul, de ahelyett, hogy végig haladnánk a mélységi keresésen egy fix távolság után mindig visszalép az algoritmus és a lehetséges tovább haladási utak közül tovább generálja a labirintust. Ezzel a megoldással több rövidebb utat generál a program és növeli a lehetséges elágazások számát, ezáltal kevesebb egyenesszakaszból áll. Az igy elkészített labirintusban minden egyes labirintus terület elérhető lesz így a játék minden esetben végig játszható lesz.

# **Modellek készítésé**

A modellek elkészítéséhez a Blender-t használtam és innen exportáltam át Unity-be FBX. ként. A modellek amiket el kellet készítenem a labirintushoz:

* a fáklyák, melyeket a falakra helyeztem
* a szobához tartozó csillár
* a játékhoz az ellenséges kutyák modelljét
* a kutyák ártalmatlanná tételéhez a csontokat
* a csontokhoz a dobozt, amelyben tárolva lesz a csont addig, amíg a játékos fel nem veszi
* a szobákhoz szükséges bejáratiajtó
* a szobákba elhelyezésre kerülő könyvespolc, asztal, szekrény , könyv, szék és az asztalra elhelyezett gyertyának a modelje

# **Fáklya, gyertya és csillár**

Ezt a három modellt egy egységbe veszem mivel ezeknek a fő szerepe a fényforrás. A modelljeik elkészítésénél igyekeztem középkori játékokhoz hasonló stílusút készíteni, ennek a korszaknak a tárgyai illenek a játék történethez.

Előszőr a falon elhelyezett fáklyákat készítettem el. Ezekhez egy elég letisztult kép volt a fejemben. Egy középkori stílusú fantasy falon lógó fáklyánál maradtam. A fáklyának a vas tartószerkezetéhez egy alacsonyabb felbontású tóruszt használtam kiindulási alapnak. A tórusz egy olyan alakzat, amelyet akkor kapunk, ha egy téglalapnak az oldalait összekapcsoljuk a velük szemközti oldallal igy létrejön egy amerikai fánk alakzat. Az elkészült fáklya modellt a 8.-as ábra mutatja be. A tóruszt ezután átalakítottam úgy, hogy az egyik felénél az oldalsó lapot meghosszabbítottam, amellyel kapcsolódni fog a falhoz. Az elkészített alakzatnak már csak hozzá kellet adni a textúráját, amivel fémes hatást tudunk elérni, majd végül a fáklyához egy kör alapú gúlát használtam, aminek az alját bemélyesztettem és meg fordítottam, hogy illeszkedjen a fáklya tartójába. A fáklyához már csak a textúráját kellet, hozzáadnom ahol a mélyedésben parázs szerű texturát hoztam létre, melyhez majd a Unity-ben hozzá adtam a particle effekteket. A gyertya és a csillár is egy kicsit összetettebb modellből áll, mint a fáklya. A gyertyához az alapzat elkészítéséhez egyetlen cilinder alakzatot használtam, amellyel az általam elképzelt gyertya és alapzata mind kör alakzatú. A cilindert felosztottam több szakaszra, majd az újonnan készült köröket átméreteztem, hogy megfelelő átmérőjűek legyenek a megálmodott gyertyához. Az elkészült gyertya modellt a 9.-es ábra mutatja. A csillár is nagyon hasonló módszerrel készült. A csillárnak a négy karjában helyezkednek el a fényt adó gyertyák. Az alapját a csillárnak egy kockából készítettem el, melynek az oldalait bővítettem és alakítottam át, majd erre csak rákerültek a gyertyák.

|  |  |
| --- | --- |
| . ábra, 7.1 fejezet: fáklya modell | . ábra, 7.1 fejezet: gyertya modell |

# **Szekrény, könyv, asztal és szék**

Az asztalhoz, a székhez és a szekrényhez egy fa színű texturát használtam és mind a négy modellt alapvetően elég egyszerű felépítésűek. A szekrény modell elkészítéséhez egy kocka modellt megnyújtottam és az egyik oldalát felosztottam egyenlő szakaszokra, majd insert faces és extrude region segítségével kialakítottam a bemélyedéseit a modellnek, ezek az elkészült szekrény modellen jól kivehetőek, melyet a 10.-se ábrán látható. Az asztal és szék elkészítése kicsit bonyolultabb volt, de a két modell hasonló alapokkal készült. Mindkettőnél a kockának az alját felosztottam négy egyenlő részre és használtam rajtuk insert faces és extrude region műveletet, hogy megkapjam a lábaikat, ez után a széknek még ki kelet alakitani a háttámla részét, melyet extrude region segítségével alakítottam ki. Azt, hogy elérjem, hogy a háttámlában legyen egy lyuk kitöröltem a hozzá tartozó két oldalt és a belsejiben meg hozzá adtam az oldalakat, hogy ne lehessen belátni a modellbe. Az asztal és a szék modelljét együtt a 11.-es ábrán mutatom be. A könyv modellje egy egyszerű átméretezett kocka és erre készítettem el a könyvnek a textúráját, melyet a 12.-es ábrán láthatunk.

|  |  |  |
| --- | --- | --- |
| . ábra, 7.2 fejezet: szekrény modell | . ábra, 7.2 fejezet: asztal és szék modell | . ábra, 7.2 fejezet: könyv modell |

# **Kutya**

A kutya modelljének az elkészítéséhez előszőr hozzá adatam a subdivision modifier-t és egy kockákból kialakítottam a kutya alakzatot. A lábaihoz azt a módszert használtam, amit már az asztalnál is használtam. A 13.-mas ábrán látszik (ahogy a subdivision módosító eredményét és) az elkészült kutya modell.



. ábra, 7.3 fejezet: kutya modell

# **Scriptek**

A Unity-ben a scriptek hozzáadásával értem el, azt hogy a játék működjön. A különböző scriptekben van meghatározva, hogy az egyes elemek mit csináljanak vagy, hogy hogyan készüljön el a labirintus. Ezek a scriptek C# nyelven vannak megírva és a Unity-nek a könyvtárát használja, hogy képes legyen kommunikálni a script a Unity-nek a motorjával. Az Én általam írt összes scriptben definiált osztály a MonoBehaviour-ből származik. Amely azért felelős, hogy az ebben kapott függvényekben tudunk a motorral érintkezni. Ilyen függvény például a Start és az Update függvény. A Start függvényt akkor fogja meghívni a motor, amikor az elindul. Ez a függvény a működés során csak egyszer fog lefutni, míg az Update methodus minden egyes frissítésben meghívásra kerül a motor által.

# **Button Manager script**

A ButtonManager script-ben létrehoztam függvényeket, amelyeket az egyes színterekben használt gombokhoz rendeltem. A gombok a lenyomásukkor meghívják a gombhoz rendelt függvényt, ilyen a játék elindításához tartozó gomb.

# **Player script**

Ebben a script-ben végzem el a játékoshoz kapcsolódó főbb műveleteket, ilyen például a karakter mozgásnak a megvalósítás. A script-ben lekérdezésre kerül a játékos jelenlegi helyzete, amelyet a játékos GameObject-től vagyok képes lekérdezni és ennek a változtatásával vagyok képes a játékos mozgatására. A Unity alapból kezel inputokat és ezzel egy egységesebb irányitást tesz lehetővé, de emellett képesek vagyunk mi is gombok lenyomását lekérdezni a motortól. A Unity-ben definiálva van egy Horizontal és egy Vertical változó, melyben a motor lekezeli, hogy beállításai alapján milyen inputok változtatják ezeket a változókat. Alapból ezeket a „w-a-s-d” gombok, a nyilak vannak beállítva, de emellett be van még állítva a joystick használatának lehetősége is, azonban mi is képesek vagyunk ezen változtatni, ha szeretnénk. Én ezen két változó alapján határoztam meg a játékos mozgását, melyet még befolyásol, hogy a karakter érintkezik-e a talajjal. Amennyiben a játékos érintkezik a talajjal akkor képes ugrani. Ezt úgy oldottam meg, hogy megvizsgáltam, hogy a lábától lefele található-e bármilyen objektum és ha igen akkor beállítottam, hogy képes legyen ugrani. Ellenkező esetben nem ugorhat a játékos. A játékos irányitó scripten helyezkedik még el az a funkció, hogy ha a játékos olyan objektumra néz, amivel képes interakcióra lépni akkor azt kiírja a képernyőre. Ha a játékos megnyomja a megfelelő gombot akkor elindítja a cselekvéshez szükséges műveleteket ilyen például, hogy oda adja a kutyának a csontot és a kutya megáll, vagy kinyitja az ajtót.

# **Torch script**

Ez a script szerepel az össze fáklyában a labirintuson, ez felelős a fáklyának a ParticleSystem kezeléséért és a fáklya hangjáért. Ha a játékos egy bizonyos távolságon belülre ér a fáklyához az bekapcsol, világít és hangot ad ki. Amennyiben a játékos eltávolodik a fáklyától, akkor az lekapcsol, ezzel is csökkenti a motor terhelését és így elősegíti a játék futását. A fáklyák fényét úgy állítottam be, hogy az intenzitása az alapján nőjön, hogy a játékos mekkora távolságra van a fáklyától, igy egy halványául és erősödő hatást elérve.

# **Dog script**

A kutyának a scriptje eltárolja a labirintus térképét, illetve eltárolja a saját helyzetét és a játékos helyzetét. A kutya önállóan mozog a labirintusban, amennyiben a játékost látja a térképen egyenes vonalban irányába halad. Ebben a scriptben végzem el, hogy a kutya megálljon egy kis időre amikor az egyik cellából áthalad a következőbe, vagy fordul, ha a játékos odaadja a kutyának a csontot akkor megnöveljük a kutya várakozásának az idejét. Minden alkalommal amikor meghívóik az Update metódusa a Dog scipnek, akkor addig vonjuk ki az eltelt időt amíg a várakozási ideje újra nem lesz nulla és haladhat tovább a kutya. A kutyának a játék elején pár másodpercig még nem indul el, ezzel előnyt adva a játékosnak. A 14.-mas ábra ebben a szakaszban készült és látható rajta, hogy a játék feltünteti, hogy nincs a játékosnál még csont melyen megállíthatja majd a kutyát.



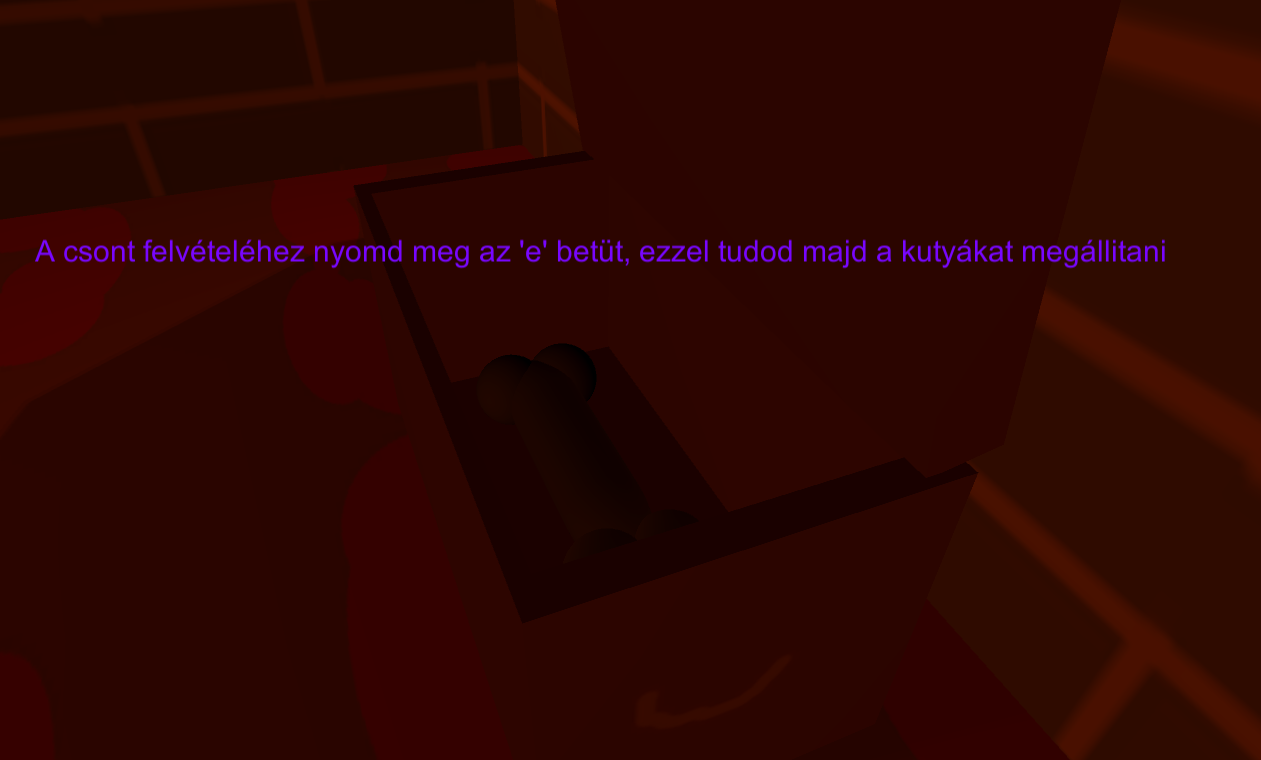
. ábra 8,4 fejezet: kutya a Unityben a játék elején

# **Ajtó script**

Az ajtó scriptjében kezeltem az animációját és mentettem el, hogy éppen nyitott állapotban vagy csukott állapotban. Mikor a játékos ránéz egy olyan ajtóra mely tartalmazza ezt a scriptet akkor a játékos scriptje lekérdezi az ajtó scriptjétől, hogy éppen nyitható-e. A játékosos scriptje csak akkor írja ki, hogy az ajtó nyitható, ha az ajtó scriptje visszaadja, hogy nyitható.

# **Láda script**

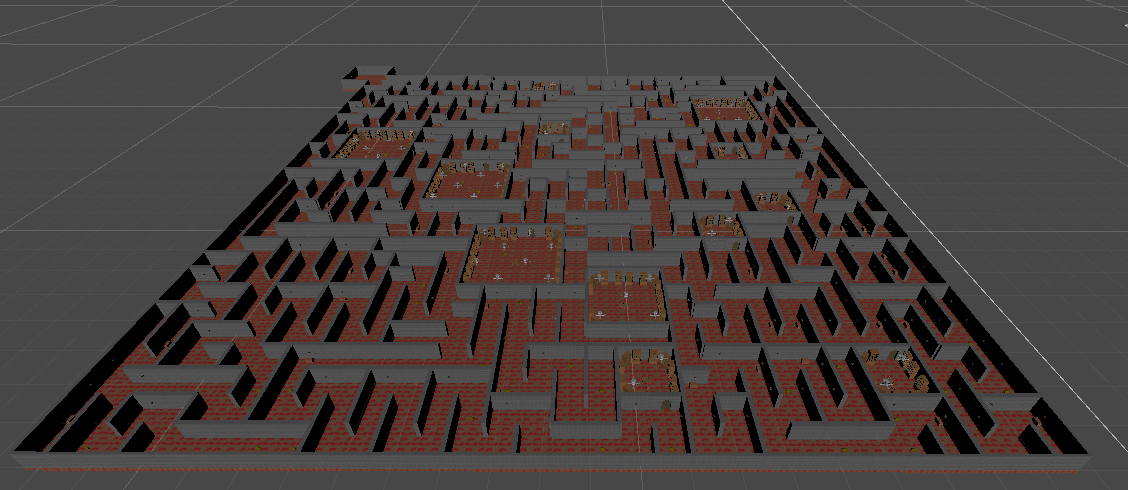
Az láda scriptjében kezeltem a láda állapotát, hogy éppen nyitott állapotban vagy csukott állapotban. Ebben a scriptben kezelem a ládához kapcsolódó animátorhoz kapcsolódó feltételeket és ezzel irányítva az animátort. Mikor a játékos ránéz egy olyan ládára, mely tartalmazza ezt a scriptet akkor a játékos scriptje lekérdezi, hogy a ládascriptjétől, hogy éppen milyen állapotban van. A játékosos scriptje csak akkor írja ki, hogy az láda nyitható, ha a láda csukott állapotban van, minden más esetben nem ír ki semmit. A 15.-ös ábrán látható, hogy a játékban a játékos kinyitotta a ládát és ezzel elérhetővé vált a benne rejlő csont.



. ábra, 8.6 fejezet: A nyitott láda a játékban benne a csontal

# **Mazegenerater script**

A mazegenerater script a játék egyik legfontosabb eleme, ebben hozom létre a labirintust és ebben adom hozzá az össze elemet a labirintushoz. Ilyen elemek: a kutyák, fáklyák és az összes elem a játékban a játékoson kívül. A script amikor elindul a Start metódusban akkor a script betölti az össze prefab objektumot, ami, majd a játékban elhelyezésre kerül. Ezután kiszámolásra kerül, hogy a labirintus milyen méretű legyen. Miután megvan a labirintus mérete, elkészül az üres labirintus és az összes ehhez szükséges segédtáblázat. Amint a táblázatok elkészültek létrehozásra kerül egy kulcsváltozó, ami a labirintus generálásának a mintáját fogja meghatározni annak érdekében hogyha tovább fejlődik a játék akkor ez a kulcs lekérhető legyen és újra lehessen játszani azokat a labirintusokat, amelyek megtetszettek a felhasználóknak. A kulcsban elmentésre kerül, hogy hol kezdődik a labirintus generálása, mekkora mélységig haladjon a mélységi keresés során az algoritmus és végül, hogy merre próbáljon haladni a mélységi keresés során az algoritmus, ha képes arra haladni. Ezekkel a paraméterekkel képesek vagyunk a kulcs és a labirintus méretei alapján azonos felépítésű labirintus generálására. Miután elkészült a kulcs, nekilátunk a tervek szerint elhelyezni a szobákat a labirintusban a generateRoom metódusban. Ezután már csak végre kell hajtanunk az algoritmust, hogy a labirintus térképe elkészüljön. Ezt követően már lehelyezhetjük az objektumokat a sceneben. Ahhoz, hogy egy objektumot lehelyezhessünk tudnunk kell, hogy hol helyezük el a háromdimenziós térben. Ehhez eltároltam a cella szélességét és egy offset-et, hogy hova helyezzem el, így a táblázat alapján már könnyen el lehetett helyezni az egyes cella elemeket. A falak elhelyezéséhez elég megnézni, hogy a táblázat mely elemét nézzük. Mivel egy háromdimenziós tömböt használtam, hogy minden egyes cellának elmentsem a négy oldalát. Elég tudnunk, a labirintust helyzetét a világ középpontjához képes és a cella szélességét. Ezeket felszorozva a táblázat két dimenziójával megkapjuk a cella középpontját. Az alapján, hogy a négy fal közül melyiket helyezzük le, elforgatjuk és arrébb csúsztatjuk a cella középpontjától. Minden egyes cellának egyik falán lennie kell egy fáklyának. A szobákban elhelyezett tárgyak helyzetét is hasonló módon kapjuk meg. Mivel minden tárgynak van valami kapcsolata egy cellával, így könnyen kiszámolható a helyzetük. Ilyen például a szekrények elhelyezése, melyből minden falnál kettőnek kell lennie. Ha ez egyik sarokban van akkor nem férne el két szekrény egymás mellett, ezért csak egy szekrény került elforgatva le helyezésre. Az asztalok mindig a szoba közepére kerülnek és a szék az egyik hosszabbik oldalára kerül. Mivel tudjuk, hogy helyezkedik el az asztal, ezáltal a random lehelyezett gyertyát is pontosan el tudjuk helyezni az asztalra. A csillárok elhelyezése a szobában lévő cellákban egy sakktábla mintát követnek. Amelyet könnyen megkaphattunk, mivel tudjuk, az alábbi képlet alapján i%2==j%2, hogy mely celláról van szó. A 16.-os ábrán egy 30x30as legenerált labirintus a látható amelyet a szemléltetés kedvéért megállítottam, levettem a menyezeteket és ezzel láthatóvá vált a labirintus felépítésé.



. ábra, 8.7 fejezet: Egy elkészült labirintus a játékban mennyezet nélkül látva a generálás végeredményét

# **Animációk**

Az általam elkészített modellek mellé az animációkat a Blender segítségével készítettem el. Az animációt a modellel együtt kimentettem az FBX fájlba és onnan be importáltam a Unitybe. Beimportálás után az animációk egy animációban szerepelnek, amelyeket az animáció hossza alapján felszeleteltem és így kimentettem külön animációkra bontva. Az animációkat az animátor segítségével lehet rendszerbe fűzni. Az animátorok működésükben egy automatára hasonlítanak, ahol az egyeses állapotokból egy hatás segítségével átkerül egy másik állapotba. Ez álltál bonyolult animátorokat hozhatunk létre. Az animátorban az egyes animáció átmenetekhez feltételeket, amelyek, ha teljesültek akkor az animátor képes átlépni a következő állapotába. Az animátorokban az animációknak és az átmeneteiknél is van lehetőségünk beállítások változtatására. Ilyen beállítás az animációnál a lejátszás sebesége vagy az átmenetnél a feltételek. Az feltételeknek négy típusuk van.

* egész értékű (integer)
* lebegőpontos (float)
* logikai (bool)
* Trigger

A feltételekhez beállíthatunk egy értéket, ami alapján a feltétel teljesülhet. A lebegőpontos számoknál be lehet állitani, hogy az adott értéknek hogyan viszonyulnia egy határértékhez. A lebegőpontos számoknál nem állíthatunk be egyenlőséget a feltételnél egy határértéke, csak hogy kisebb vagy nagyobb-e. Az egész értékű feltételnél beállíthatjuk, hogy kisebb, nagyobb, egyenlő vagy nem legyen egyenlő egy határértékkel. A logikai változónál határérték helyett azt vizsgálhatjuk meg hogy a logikai változó igaz vagy hamis-e. A trigger-nél maga a változó számít a feltételben, ilyen lehet egy gomb lenyomása. Ezeket a feltételeket össze is lehet fűzni, hogy bonyolultabb feltételeket kapjunk. Így akár intervallumokra is rá tudunk szűrni a feltételek során. A játékban négy animációt hoztam létre, ezek közül kettőt a Blenderben készítettem el és importáltam a Unitybe, a másik kettőt a Unityben. Az animációkat melyeket a Blenderben készítettem azokat a csontok segítségével készítettem el. Azokat az animációkat készítettem el a Unityben amelyeknél a modell helyzetét és a forgását változtattam.

* Játékos animációja
* Kutya animációja
* Láda animációja
* Ajtó animációja

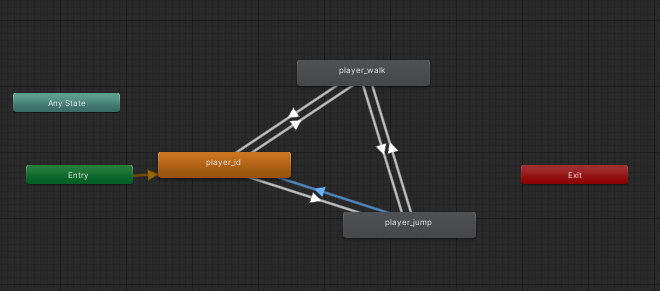
Ezeket a következőkben részletesen kifejtem.

# **Játékos animációja**

Ebben az animátorban három animáció szerepel. Ezek az animációk a:

* Idle, mikor csak áll a karakter
* Run, mikor a szaladás történik
* Jump amikor az ugrás animáció játszódik le

Ezekből az animációkból mindegyik után képes következni bármelyik. Ezáltal ekörül a három közül bármelyik állapotból képes átkerülni az animátor bármelyik másik állapotba. Ebben az animátorban kettő logikai változót használtam a feltételek során. Egyet amikor a karakternek ugrania kell és egyet amikor futnia. A változókat a játékosnak a scriptjében állítottam be a megfelelő állapotukba. Az ugrás változóját akkor állítom be igazra amikor a játékos megnyomja a megfelelő gombot és a karakter ugrik. Szóval, ha a karakter egyszerűen elkezd zuhanni ugrás nélkül akkor nem fog lejátszódni az ugrás animációja, mivel a karakter nem ugrót és akkor állítom vissza hamisra ismét mikor a talajra érkezet. A futás értékét ugyan ilyen módszerrel változtatom. Az elkészült animátort a 17.-es ábrán látható.



. ábra, 9.1 fejezet: A játékos animátora

# **Kutya animációja**

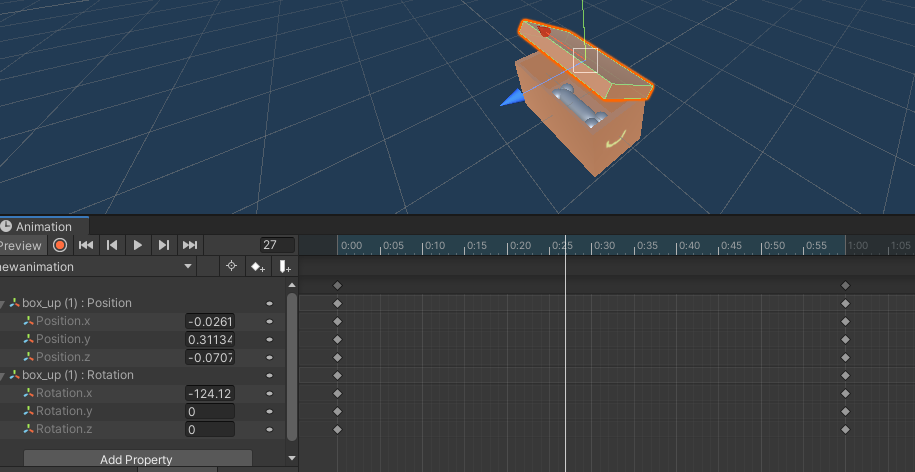
A kutya animációjában a sétálás szerepel. Ezt az animációt a kutya scriptjében állítom meg és indítom el az animátoron keresztül. A kutya a játék elején vár 10 másodpercet és csak utána indul el. A kutyának a sétáló animációját a fordulás során nem állítom meg.

# **Láda animációja**

A ládának az animátorában három animációt használtam.

* csukott állapotban
* a nyitás animációja
* nyitott állapotban

Azért volt szükségem három állásra, hogy a láda amig a játékos nem interaktál vele addig csukva maradjon és a kinyitás elvégzése után ne ugorjon vissza az animáció előtti állapotba A láda animációját a Unityben hoztam létre, mert ennek a tárgynak a meganimálásához szükséges volt mozgatni és forgatni a tárgyat. A mozoggatás és forgatás animációjának beállítását a 18.-as ábra mutatja be. De nem volt szükséges csontok használata mivel az alját és a tetejét külön egységből építettem fel. Könnyebb az elhelyezkedéseket a Unityben átlátni, hogy hogy lesz a leg megfelel előbb a játékban és gyorsabb volt kipróbálni mivel az exportlásás és importálás rész kihagyhattam ez által, hogy a Unityben készítettem el. A felnyitás feltételéhez egy logikai változót hoztam létre, amelyet akkor állítottam igazra amikor a játékos rá kattintott a kinyitásra. Az animátor a csukott állapotból a nyitás hatására átkerül a nyitás animációba. Miután a nyitás animáció befejeződött átkerül a nyitott állapotba.



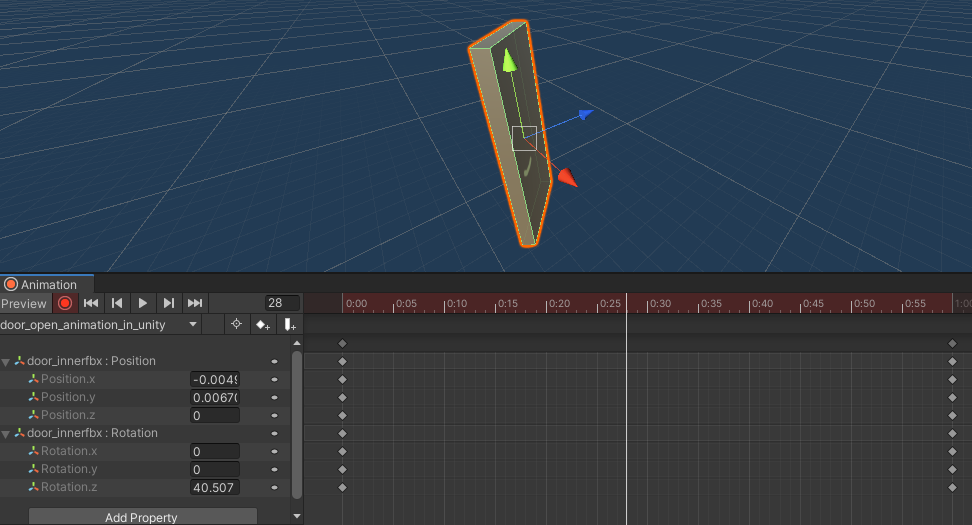
. ábra, 9.3 fejezet: A doboz nyitódásának animációjának egy pillanata

# **Ajtó animációja**

Az ajtó animáció hasonlóan készült, mint a ládának az animációja. Szükségem volt hozzá három animációra.

* csukott állapotban
* a nyitás animációja
* nyitott állapotban

A csukódás animációjára a nyitás animációjának a lejátszását átállítottam negatívra ezáltal egy egységesebb animáció jön létre. Az animátornál négy állapotot állítottam be. Ezt egy kör alakzatban rendeztem el. Az animációk kezeléséhez két triggert használtam, egyet a nyitás animáció indításához és egyet a csukás animáció indításához. Miután a csukás vagy a nyitás animáció befejeződött átkerül a nyitott állapotába vagy a csukott állapotába. A csukódás animációját hasonlóan a Unityben készítettem el, az animációnak az értékeinek beállítását mutatja be a 19.-es ábra.



. ábra, 9.4 fejezet: Az ajtó nyitódásának az animációjának egy pillanata

# **Unity prefabok**

A Unityben a prefabok olyan GameObject-ek melyeket mi hozunk létre már létező GameObjectek-ből. A prefabok olyan eszközök melyeket egyszerüen be tudunk helyezni a szintérben és az eredeti prefab tulajdonságaival fognak rendelkezni. A prefabokat készíthetünk más prefabokból is ezzel bonyolultabb egységeket létrehozva. Azokat a GameObjecte-ket melyeket prefab-bá szeretnénk alakítani, azokat egyszerűen megfogjuk és a fájl rendszer egyik mappájába helyezük. A prefabokat általában a Resources mappán belül egy prefab mappába szokás helyezni az elkülönítés érdekesben. A prefabok hasznossága hogyha változtatunk a prefabon akkor az annak lévő össze másolata is automatikusan vele fog változni. A prefabok az apró építő elemei a játékoknak. A játékban szinte minden elem, ami a szintéren letételre kerül az egy prefabol származik. Ilyen például

* A kutyák
* a falak
* a padló és a mennyezet
* a csont

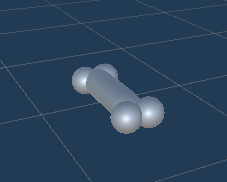
A játékban több olyan prefabot is használok, amiben egy másik prefab is szerepel ilyenek

* A falak, melyeken fáklya szerepel
* A fal, amelyen van az ajtó
* A mennyezet, melyen lóg a csillár
* A láda, melyben a csont van

A következőkben kitérek pára ezek közül

# **Csont prefab**

A csont prefabja tartalmazza négy gömböt, egy kapszula alakzat és az azokhoz tartozó collidereket. A csont ezekből az alakzatokból épül fel, mely a 20-as ábrán látható. A csontnak van egy hasonló scriptje, mint az érmének, amely azért felel, hogy a fel és le lebegjen.



. ábra, 10.1.fejezet: A csont prefab-ja a Unityben

# **Láda prefab**

A láda prefab-jában szerepel a ládának a tetejének a modellje, a láda aljának a modellje és a csontnak a prefabja. A ládának minden oldalán van egy collider mely megakadályozza a játékost, hogy átsétáljon a dobozon. A doboz tetején szerepel a már korábbiakban tárgyalt Chest script. Mely majd elvégzi a láda tétjééhez tartozó animátornak szükséges feltételek kezelését. Az animáció lejátszódása után és előtti állapotot demonstrálják a 21.-es és a 22.-es ábra. Ezt a doboz prefabot a labirintust generáló script fogja elhelyezni a labirintusban az alapján, hogy milyen gyakori a láda a labirintusban.

|  |  |
| --- | --- |
| . ábra, 10.2 fejezet: Unityben a láda prefab nyitott állapotban | . ábra, 10.2 fejezet: Unityben a láda prefab csukott állapotban |

# **Kutya prefab**

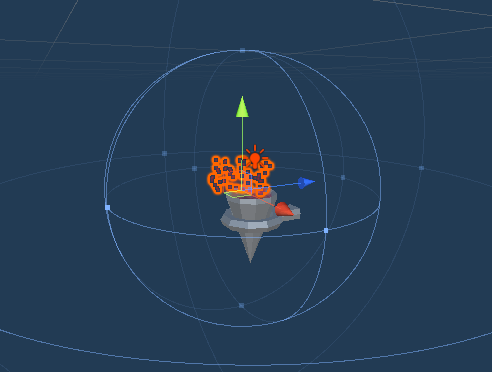
A kutya prefabja tartalmazza a korábban Blenderben elkészített modellt és a hozzá tartozó animátort. Ebben a prefab-ban még természetesen szerepel a kutyának a scriptje és a scriptekhez tartozó két fényforrás. Ha lekapcsolnak a fények a játékos akkor is tudja majd, hogy hol található az ellenséges kutya, amely felé tart. A 23.-as ábrán látható a prefab-ban a kutyának a fényforrása és a modellje. Ezt a prefabot is a labirintust generáló srcipt helyezi el a labirintusban. A játék elején a kutya mindig vár egy kis időt mielőtt elindul.



. ábra, 10.3 fejezet: Unityben a kutya prefabja, melyen látszódnak a szeménél elhelyezett két fényforrás

# **Fáklya prefab**

A fáklya prefab-ba elhelyeztem a fáklyának a modelljét a megfelelő beállításokkal, mint például a hozzá tartozó materiál, árnyék. A fáklyához hozzá adtam még a szüksége fényforrást, melyhez a hagyományos Point lightot választottam. A fénynek egy narancssárgás vöröses színt állítottam be, hogy megfelelő tűz hatása legyen. A prefabhoz még hozzá adtam egy Particle System, melyet úgy állítottam be, hogy egy széles kúp alakzatban gyorsan engedje kifele a részecskéket, melyek csak rövid ideig maradnak meg evvel a tűz pattogó érzetét keltve. A prefabon a fényt, a részecske rendszert és a hangforrás határát a 24.-es ábra szemlélteti. Hogy elősegítsem a ropogó tűz érzetét beállítottam az objektumnak egy hangforrást, melyből pattogó tűz hallatszik, ha a játékos megfelelő távolságba kerül hozzá.

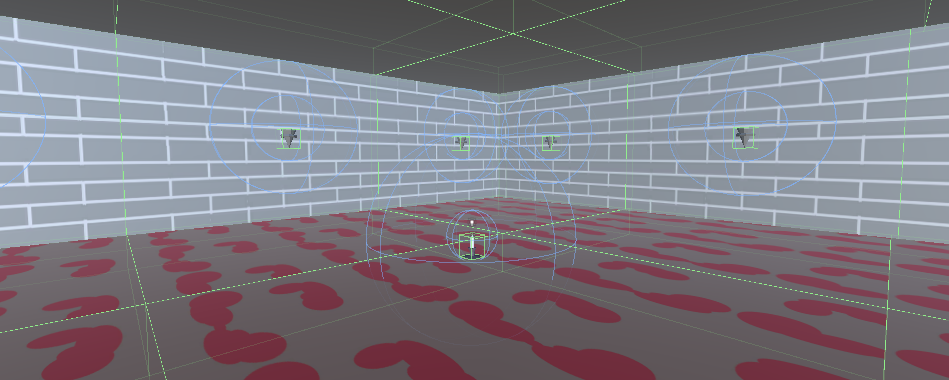


. ábra, 10.4 fejezet: Unityben a fáklya prefabja, melyen látszódik a prefab-nak az egyes elemei

# **Utolsó szoba prefab**

Az utolsó szobára készített prefab egy háromszor hármas szoba nagyságú terület. Ahova, ha a játékos elér, akkor elhagyhatja a labirintust. Ezen prefabot az eddig használt prefabok segítségével építettem ki, melyhez még hozzá adtam egy collidert, amit beállítottam trigger módra. Ha a játékos belép erre a területre akkor képes legyen elhagyni a labirintust és kiíródjon a megfelelő szöveg. Ezen a területen már a kutyák sem követik a játékost. Ehhez a prefabhoz azt a fal prefabot használtam, melyen a fáklya is szerepel, hogy a lehető legkivilágítottabb legyen és sima padló és plafon elemeket használtam hozza. Végül a szoba középre lehelyeztem egy gyertyát, hogy a játékos könnyen észre vegye, hogy ez a keresett kijárat.

A 25.-ős ábra mutatja meg az utolsó szoba felépítését.



. ábra, 10.5 fejezet: Unityben az utolsó szobának a prefabja feltüntetve rajta a benne szereplő tárgyakat

# **Nyilatkozat**

Alulírott Galgóczki Norbert programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus 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 Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2021. május. 13.

aláírás: ……………………………………….