Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Galgóczki Norbert

2021

Szegedi Tudományegyetem

Informatikai Intézet

Három dimenziós labirintus játék megvalósítása Unityben

SZAKDOLGOZAT

Készítette:
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 Unity-ben

- A megadott feladat megfogalmazása:
  - Unity-ben 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 labirintust, majd az irányító scriptekkel játszhatóvá tettem a játékot.

- Alkalmazott eszközök, módszerek:
  - A játék készítésé során a Unity által kapott eszközöket használtam a játék elkészítéséhez és Blender-t használtam a modellek és animációk elkészítéséhez, amit importáltam a Unity-be

- Elért eredmények:
  - Játszató verzió elkészült és fel van téve publikusan GitHub-ra.

- Kulcsszavak:
  - Unity, Blender, Labirintus, Játék, Generáló algoritmus

Tartalomjegyzék

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 felhasználó 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, 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. Majd ezek után adtam hozzá a játékost, aki be tudja járni a scene-térben 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. A kezdőlapon kapott helyet a beállítások felület, ahol be lehet állitani a játék hangerősségét és az egér érzékenységét.

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

1. Unity

1.1. Játékmotorok

A játékmotorok azért készülnek, hogy segítsék a játékfejlesztő munkáját a játékprogramok készítésénél. A motorok már tartalmaznak olyan alapfunkciókat, melyek mindig felmerülnek egy játék készítése 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.
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 fejlesztettek 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 készítők minél szélesebb körét ki tudják elégíteni. Ezek a motorok a lehető legtöbb fájlt képesek kezelni és a lehető legtöbb dologban tudják segíteni a fejlesztőket a játékuk készítésében. Kapcsolódó irodalmi jegyzék [5]

1.2. 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, 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. kiterjesztést. A Unity a fény kezelésnél is rengeteg grafikai beállítást és lehetőséget biztosít a játéktervezők részére.

2. Blender

2.1. 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 a 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 spline-n függvények görbéit 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**. A primitív modellt más programok csak szerkesztési eszközként alkalmazzák, az így generált objektumot alakítják később poligonhálóvá. Kapcsolódó irodalmi cikkek [2,7]

2.2. Blender alapok

Az egyik legnépszerűbb modellező program a Blender, melyről az egyetemi tanulmányom során részleges ismereteket szereztem. Az egyetemen az alap irányitásokat vettük át, de a modellező program 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 menedzselni tudjuk 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
3) egy kamerát, amely az alakzat felé néz

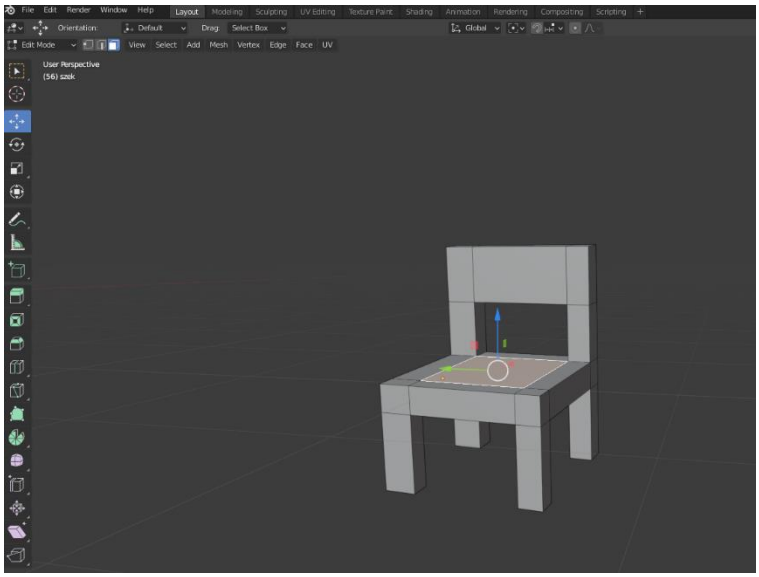
Az alakzat formázása oly módon történik, hogy az új alakzat pontokból áll és ezeket a pontokat szélek kötik össze. A szélekből lesznek az oldalak ezeket hívjuk Vertex-nek, Edge-nek és Face-nek, melyekre később hivatkozni fogunk.

A Blender-ben több nézet is elérhető, melyek a különböző célok elérésére szolgának és segítik az alkotót. Főbb ablakok közé tartozik az Object mód, melyben az objektumainkat tudjuk menedzselni. Ebben a módban 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 méréseket 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 mód, ahol az object módban kiválasztott alakzatot tudjuk átalakítani. Az Editor módban 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:

a. Extrude Region, ahol attól függően, hogy Vertex-et vagy Edge-t vagy Face-t oldalt jelöltünk ki, azt fogja kiemelni.

b. az Insert Faces, ahol a kijelölt Face-ek alapján egy új face-t hoz létre

c. a Bevel, amely a kijelölések alapján a kurzor irányításával kivág egy szeletet az alakzatból

d. a Loop cut, ami az alakzat körül egy hurkot képez és a hurok mentén elvágja az alakzatokat.

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

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

Ezek a következők:

– az UV Editor mód

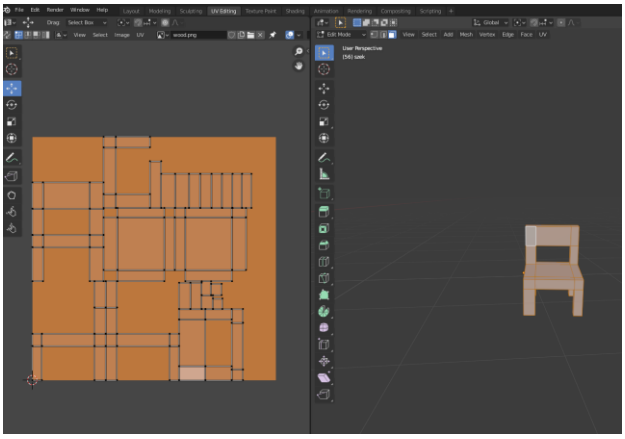
– a Textura painter mód

– Animation mód

Ezekkel a módokkal tudtam az elkészült alakzatok textúráját az-az kinézetét alakítani és véglegesíteni. Az UV Editor 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, a forgatás és az átméretezés eszközünk az UV Editor-ban is rendelkezésünkre áll. Ezek segítségével tudjuk a térképen elhelyezni a Face-ek Vertex-eit, hogy azoknak 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 a 3D alakzatunkat miképpen hajtsa szét, hogy az kétdimenzióson elférjen az UV térképen. Ahogyan a kisgyerekek a 3D kockát hajtogatják egy papírlapból, mi is be tudjuk jelölni a vágás helyét az alakzatunkon. Ezt a lehetőséget a 2-es ábra mutatja be.
- vagy beállíthatjuk azt, hogy a 3D alakzatunkat úgy vetítse le az UV térképre, ahogy azt mi látjuk a képszerkesztőben



2. ábra, 2.2 fejezet: feldarabolt szék modell

Mind a három megoldásnak megvan az előnye, 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 alapfunkciókkal rendelkező eszközöket, melyek elégségesek a Textura elkészítéséhez. Azonban a beépített eszközök nem professzionális képszerkesztő eszközök.

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

Ezt a Weight Paintel lehet alakítani. 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 módban rögzíthetjük.

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

3. Fényforrások

3.1. Fényforrások alapjainak ismertetése

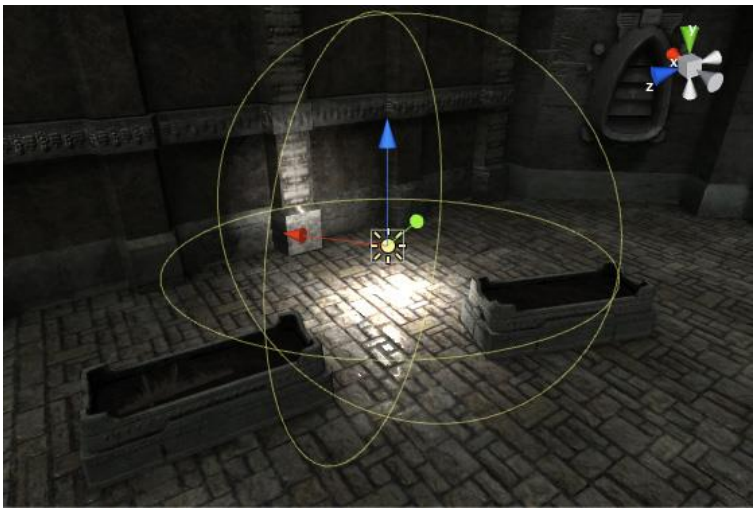
A játékoknak és a modellezésnek egyik nagyon fontos eleme a fény és a fénnyel való játék.
A fénynek és a fényforrásoknak minden esetben van erőssége és színe. A legalapvetőbb fény, ami nem sorolható a fényforrások közé, de a modellezésnél elég fontos, az az Ambient fény, ami egy környezeti fény. Egy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

– a Point light-ot
– a Spotlight
– a Directional light
– a Area light

Ezeknek már van különlegeségük, mint például van irányuk, van forrásuk.
A Unity-ben és a Blender-ben is ezeket használjuk. A fényforrásokhoz kapcsolódó irodalmi jegyzék [9].

3.2. Point light

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

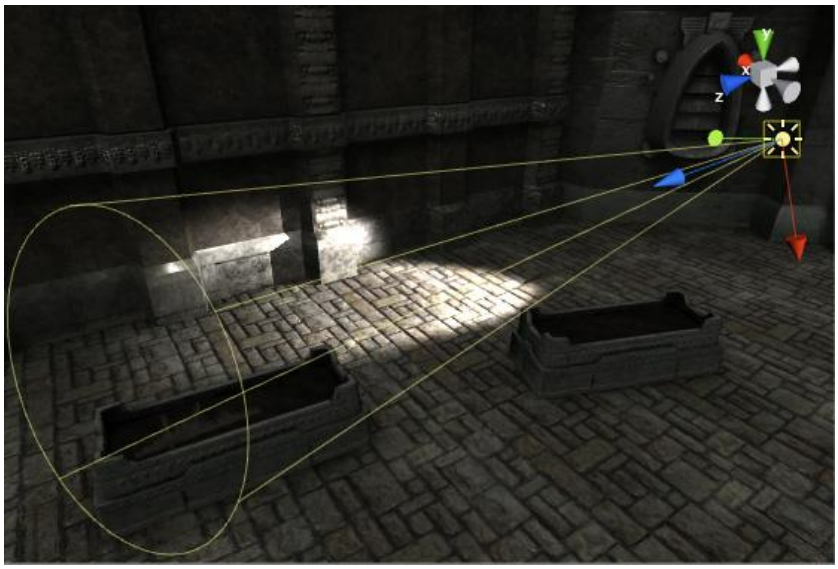


3. ábra, 3.2 fejezet: Unity Point light. Az ábra a [1] irodalomból származik

A fény, ahogy távolodik a forrásától veszít az intenzitásából. A fényintenzitás fordítottan arányos a forrástól mért távolság négyzetével. Ezt „inverz négyzet törvénynek” nevezik és hasonlít 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.

3.3. Spot light

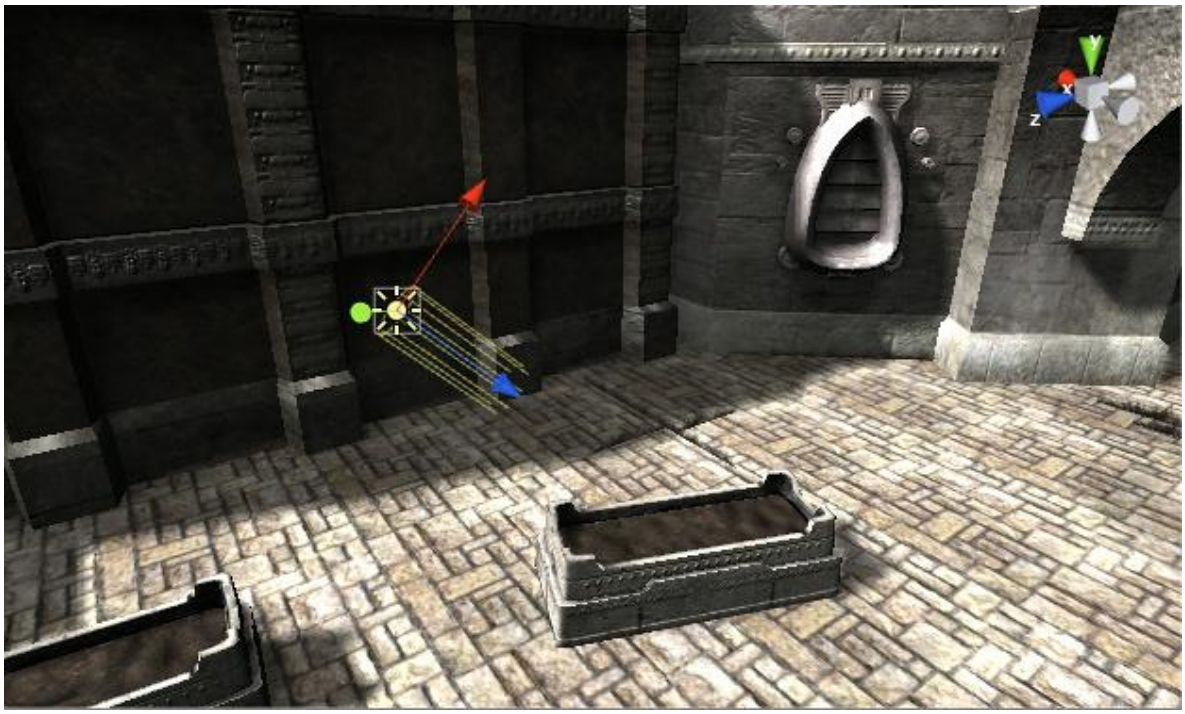
A Spot light nagyban hasonlít a Point light-hoz, rendelkezik minden Point light jellemzővel.
A fény erősségével, a fény színével, annak térbeli helyzetével, azzal, hogy milyen messzire ér el a fény és ezeken felül a Spot light rendelkezik még egy szög jellemzővel is. A szög jellemző azt jelenteni, hogy 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 fényerősség. Minél nagyobb a fényforrás szöge, annál jobban mosódik el a fény a Spot light kúpjának a szélén, ezt az angol terminológiában „penumbra” -nak nevezik. A Spot light fényforrást általában reflektoroknál, zseblámpáknál, illetve kamerák fényénél szokták alkalmazni. A 4. ábrán a Spot light fény megjelenítése látható.



4. ábra, 3.3 fejezet: spot light fény. Az ábra a [1] irodalomból származik

3.4. Directional light

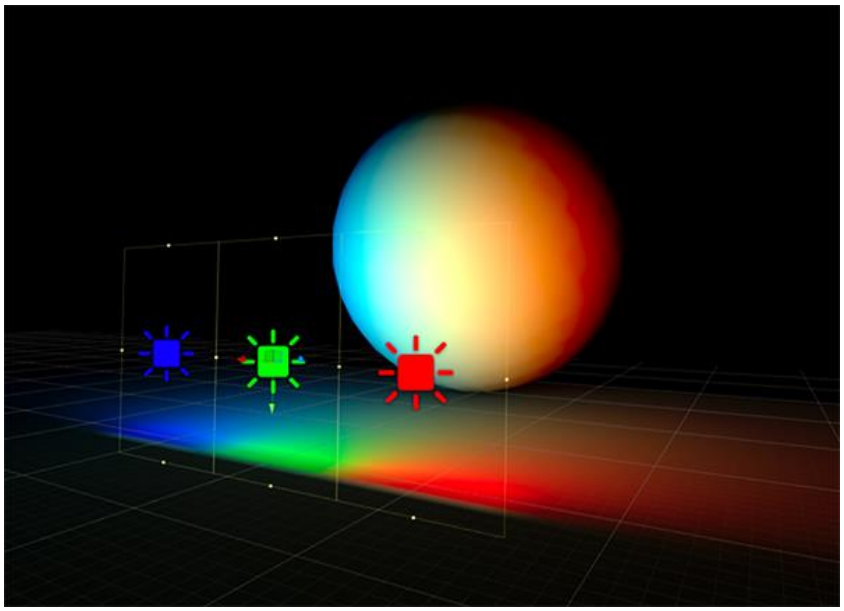
A Directional light egy alapvető fényforrás. Amikor létrehozunk egy új Unity 5 vagy magasabb verziószámú projektet, abban szerepel egy Directional light objektum. Ezt a fajta fényforrást Blender-ben Sun light-ként lehet megtalálni. A Directional light, olyan helyzetekben hasznos amikor egy nagyon távoli fényforrást szeretnénk elkészíteni, mint például 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. A fényforrás távolsága az objektumtól nincs meghatározva, ezért a fény ereje nem változik, 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.



5. ábra, 3.4 fejezet: Unity Directional light. Az ábra a [1] irodalomból származik

3.5. Area light

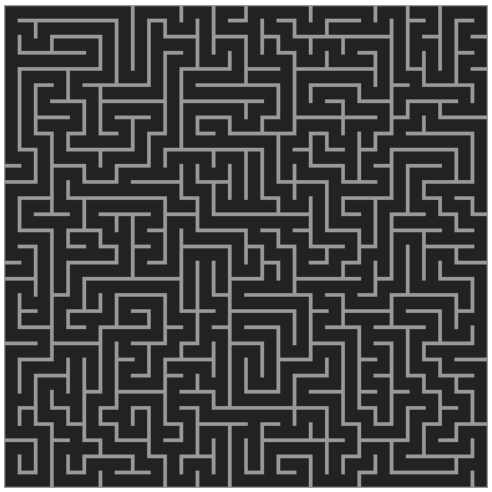
Az Area light-ot általában a háromdimenziós térben lévő téglalapként szokták ábrázolni. Az Area light ennek a téglalapnak az egyik oldalából bocsátja ki a fényét, mely minden irányba terjed, akár csak a Point light fénye. Ennek a fényforrásnak a fénye a távolság négyzetével fordítottan arányosan veszít erősségéből. Az Area light-ra a 6.-os ábra mutat példát.



6. ábra 3.5 fejezet: Unity Area light. Az ábra a [1] irodalomból származik

4. Labirintusok

A labirintusok, olyan építmények vagy objektumok, melyek tartalmaznak egy vagy több bejáratot és egy vagy több célt, ami lehet akár a kijárat is 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 azokat az egyszerű sémákkal már nem lehet könnyen kijátszani, mint például azzal, hogy mindig jobbra kell fordulnunk. A Labirintusok rengeteg stílusban léteznek. A 7.-es képen egy egyszerű hétköznapi labirintus ábráját mutatom be. A labirintusokhoz kapcsolódó irodalmi cikk [3,8].



7. ábra, 4. fejezet: Egyszerű labirintus

5. 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 felosztható 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 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.

6. 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 lennie kell szobáknak. Amennyiben a labirintust generálnám le először és abba

helyezném be a szobát, akkor olyan útvonalak alakulnának ki, amiket a játékos nem képes elérni, mert el vannak zárva. Minél több szobát generálunk a labirintusba akkor annál nagyobb valószínűséggel válna kijátszhatatlanná a labirintus. Ezért először az elkészítendő 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 labirintusnak minden egyes terület elérhető lesz, így a játék minden esetben végig játszható lesz.

7. Modellek készítése

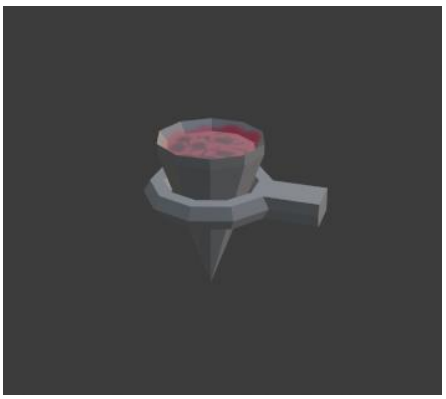
A modellek elkészítéséhez a Blender-t használtam és innen exportáltam át Unity-be fbx ként. A modellek, melyeket készítenem kellett a labirintushoz:

- a fáklyák, melyeket a falakra helyeztem
- a szobához tartozó csillárt
- 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 bejárati ajtót
- a szobákba elhelyezésre kerülő könyvespolcot asztalt, szekrényt, könyvet, széket és az asztalra elhelyezett gyertyának a modelljét

7.1. Fáklya, gyertya és csillár

Ezt a három modellt egy egységbe veszem mivel ezeknek a főszerepe a fényforrás.
A modelljeik elkészítésénél igyekeztem a középkori játékokhoz hasonló stílusú fényforrást készíteni. Ennek a korszaknak a tárgyai illenek a játék történethez.
Először a falon elhelyezett fáklyákat készítettem el. Ezekhez egy elég letisztult kép volt a fejemben. Egy középkori stílusú fantasy falon lévő fáklyánál maradtam. A fáklyának a vas tartószerkezetéhez egy alacsonyabb felbontású tóruszt használtam kiindulási alapnak. A tórusz egy olyan alakzat, melyet 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 alakzathoz már csak hozzá kellet adni a textúráját, amivel fémes hatást tudunk elérni. 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 kelleti, hozzáadnom, ahol a mélyedésben parázs szerű texturát hoztam létre, melyhez a Unity-ben hozzáadtam a particle effekteket.
A fáklyához képest a gyertya és a csillár egy kicsivel összetettebb modell. A gyertyához az alapzat elkészítéséhez egyetlen cilinder alakzatot használtam, mivel az általam elképzelt gyertya és alapzata mind kör alapú. 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 és erre csak rákerültek a gyertyák.



8. ábra, 7.1 fejezet: fáklya modell

9. ábra, 7.1 fejezet: gyertya modell

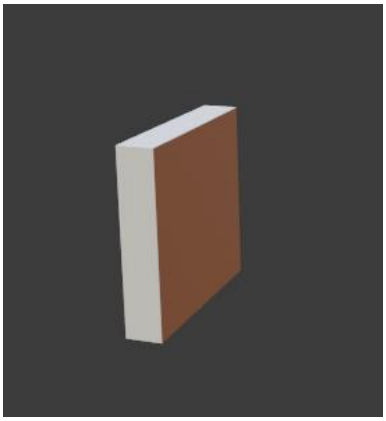
7.2. Szekrény, könyv, asztal és szék

Az asztalhoz, a székhez és a szekrényhez egy faszínű texturát használtam és mind a négy modellt alapvetően egyszerű felépítésűnek terveztem. A szekrény modell elkészítéséhez egy kocka modellt megnyújtottam és az egyik oldalát felosztottam egyenlő szakaszokra, majd az insert faces és extrude region segítségével kialakítottam a modell bemélyedéseit. A bemélyedések az elkészült szekrény modellen jól kivehetőek, mely a 10.-es 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. Ezek után a széknek még ki kellett alakitani a háttámla részét, melyet az extrude region segítségével formáltam meg. Azért, hogy a háttámlában legyen egy lyuk, kitöröltem a hozzá tartozó két oldalt és a belsejében 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.

10. ábra, 7.2 fejezet: szekrény modell

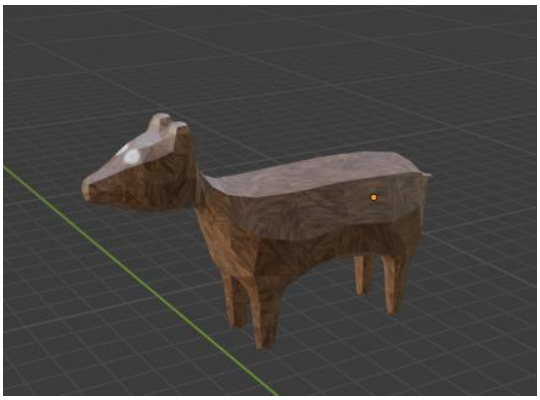
11. ábra, 7.2 fejezet: asztal és szék modell



12. ábra, 7.2 fejezet: könyv modell

7.3. Kutya

A kutya modelljének az elkészítéséhez először a kocka alakzathoz hozzáadtam a subdivision módosítót, majd kialakítottam a kutya formáját. A kutya lábait az asztalnál már használt módszer alapján készítettem el. A 13.-as ábrán látszik a subdivision módosító eredménye az elkészült kutya modellen.



13. ábra, 7.3 fejezet: kutya modell

8. Scriptek

A Unity-ben a scriptek hozzáadásával érhető el azt, hogy összetettebb cselekvések történjenek. 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 könyvtárát használják. A könyvtár használatával képes a script kommunikálni a Unity motorjával. Az általam írt összes scriptben definiált osztály a MonoBehaviour-ből származik, mely azért felelős, hogy az ebből származó függvényekkel tudunk a motorral érintkezni. Ilyen függvény például a Start és az Update függvény. A Start függvényt akkor fogja meghívni a motor, amikor a script elindul. Ez a függvény a működés során csak egyszer fog lefutni, míg az Update függvény minden egyes frissítésben meghívásra kerül.

8.1. Button Manager script

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

8.2. Player script

Ebben a script-ben végzem el a játékoshoz kapcsolódó főbb műveleteket, ilyen például a karakter mozgásnak a megvalósítása. 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 a játék irányításához a „w-a-s-d" gombok, a nyilak vannak beállítva, de emellett a joystick használatára is van lehetőség. 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, ha igen akkor beállítottam, hogy képes legyen ugrani. Ellenkező esetben nem ugorhat a játékos. A játékosirányitó scripten helyezkedik még el az a funkció, hogy ha a játékos olyan objektumra néz, amivel képes interakcióra, akkor azt kiírja a képernyőre. Ha a játékos megnyomja a megfelelő gombot, akkor elindítja a cselekvéshez szükséges műveleteket. Ilyen művelet például az, ha a játékos odaadja a kutyának a csontot, akkor a kutya megáll, vagy ha a játékos kinyitja az ajtót.

8.3. 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 könnyebb futását. A fáklyák fényét úgy állítottam be, hogy az intenzitása az alapján nőjön, hogy a játékos mekkora távolságra van a fáklyától, igy halványuló vagy erősödő hatást elérve.

8.4. 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 írtam meg azt, 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 a Dog scriptnek meghívódik az Update metódusa, akkor addig vonjuk ki az eltelt időt amíg a várakozási ideje újra nulla nem lesz és haladhat tovább a kutya. A kutyák a játék elején

néhány másodpercig várakoznak, nem indulnak el, ezzel előnyt adva a játékosnak. A 14.-es ábra ebben a szakaszban készült és látható rajta, hogy a játék feltünteti, hogy nincs a játékosnál csont mellyel megállíthatja a kutyát.

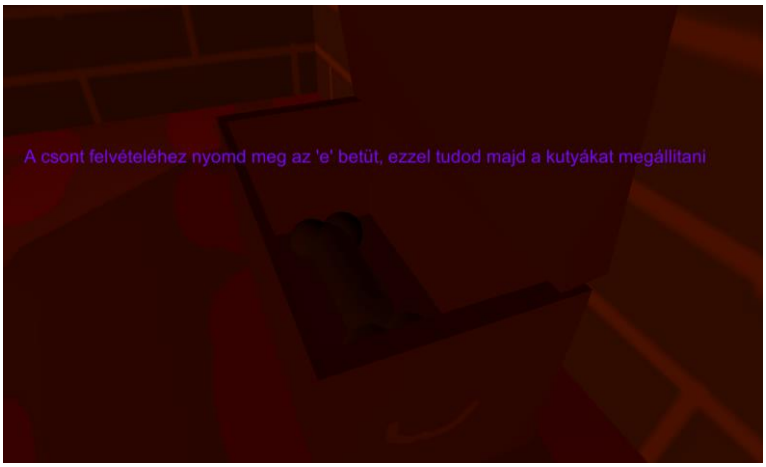
14. ábra 8,4 fejezet: kutya a Unity-ben a játék elején

8.5. Ajtó script

Ebben a script-ben kezeltem az ajtó animációját és mentettem el nyitott vagy csukott állapotát. Mikor a játékos ránéz egy olyan ajtóra, amely 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.

8.6. Láda script

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

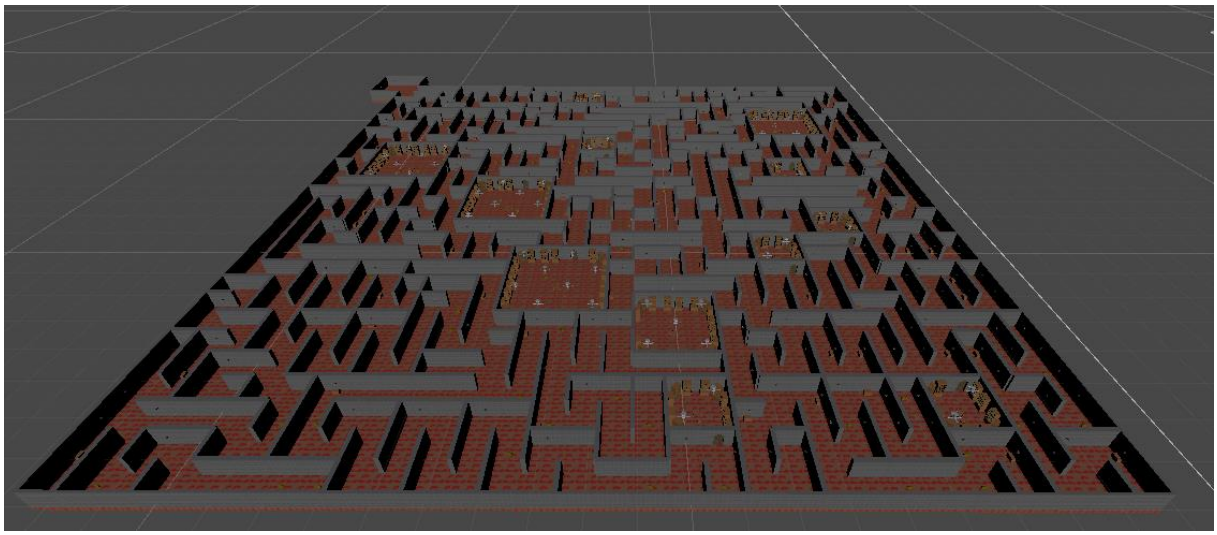


15. ábra, 8.6 fejezet: A nyitott láda a játékban benne a csonttal

8.7. Mazegenerater script

A mazegenerater script a játék egyik legfontosabb eleme, ebben a script-ben hozom létre a labirintust és adom hozzá az össze elemet. Ilyen elemek: a kutyák, fáklyák és az összes játékoson kívüli elem. A script amikor elindul a Start metódusban akkor 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, hogy a játék tovább fejleszthető legyen. Ezzel a kulccsal újra generálhatók azok a labirintusok, melyek megtetszettek a felhasználóknak. A kulcsban elmentésre kerül, hogy hol kezdődik a labirintus generálása, mekkora mélységig halad a mélységi keresés és végül, hogy merre próbáljon haladni a mélységi keresés során az algoritmus. 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 labirintusba a generateRoom metódusban. Ezután már csak végre kell hajtanunk az algoritmust, hogy elkészüljön a labirintus térképe. Az elkészült labirintusba lehelyezzük az objektumokat a scene-ben. Ahhoz, hogy egy objektumot lehelyezhessünk a labirintusba, tudnunk kell, hogy az hol helyezkedik el a háromdimenziós térben. Ehhez eltároltam a cella szélességét és egy eltolás értéket, melynek segítségével helyezzem el az objektumot. Így a táblázat alapján már könnyen el lehet helyezni az egyes cella elemeket. A falak elhelyezéséhez elég megnézni, hogy a táblázat mely cellájáról van szó. 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, a falakat 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 lehelyezésre. Az asztalok mindig a szoba közepére kerülnek, a szék pedig az asztal egyik hosszabbik oldalára. Mivel tudjuk, hogy hol helyezkedik el az asztal, ezáltal a random lehelyezett gyertyát is pontosan le tudjuk helyezni az asztalra. A csillárok elhelyezése a szobában lévő cellákban egy sakktábla mintáját követik, melyet könnyen megkaphattunk, mivel tudjuk az alábbi képlet alapján i%2==j%2, hogy mely celláról van szó. A 16.-os ábrán egy 30x30-as legenerált labirintus látható, melyben a szemléltetés kedvéért a játékot megállítottam és eltávolítottam a menyezeteket, ezzel láthatóvá vált a labirintus felépítésé.



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

9. Animációk

Az általam elkészített modellek mellé az animációkat a Blender segítségével készítettem el. Az animációt a modellel együtt kimentettem fbx fájlba, amit importáltam a Unity-be. Az importálás után az animációk egybe fűzve szerepelnek, melyeket 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 egyes állapotból egy hatás segítségével átkerül egy másik állapotba. Ezálltál bonyolult animátorokat hozhatunk létre. Az animátorban az egyes animáció átmenetekhez kapcsolódó feltételek teljesülésekor az animátor képes átlépni a következő állapotába. Az animátorokban az animációknál és az átmeneteiknél is van lehetőségünk a beállítások változtatására. Ilyen beállítás az animációnál a lejátszás sebessége vagy az átmenetnél a feltételek. Az feltételeknek négy típusa 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 kell viszonyulnia egy határértékhez. A lebegőpontos számoknál nem állíthatunk be határérték egyenlőséget, csak azt, hogy kisebb vagy nagyobb legyen. Az egész értékű feltételnél beállíthatjuk, hogy kisebb, nagyobb, egyenlő vagy ne legyen egyenlő egy határértékkel. A logikai változónál a határérték helyett azt vizsgálhatjuk meg, hogy a logikai változó igaz vagy hamis. A trigger-nél maga a változó számít a feltételbe, 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 Unity-be. A másik kettőt a Unity-ben készítettem el. A Blender-ben készített animációkat a csontok segítségével készítettem el. Azokat az animációkat készítettem el a Unity-ben, amelyeknél a modell helyzetét és a forgását változtattam.

- Játékos animációja
- Kutya animációja
- Láda animációja
- Ajtó animációja

Ezeket a következőkben részletesen kifejtem.

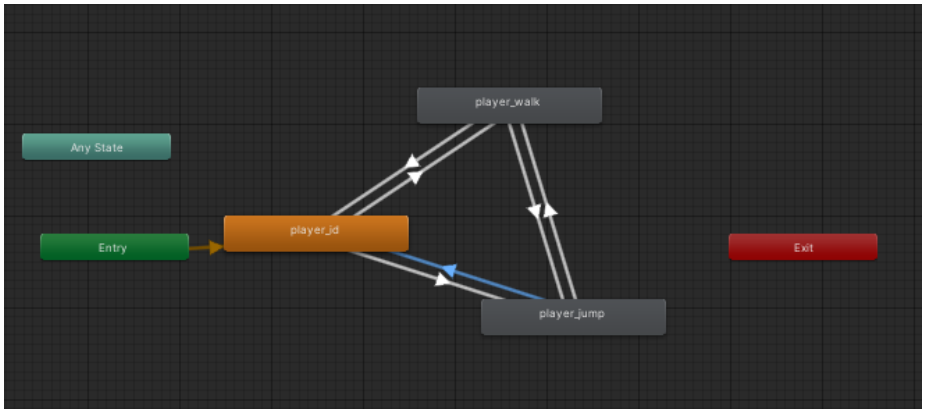
9.1. Játékos animációja

Ebben az animátorban három animáció szerepel.

Ezek az animációk:

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

A három animáció 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. Akkor állítom vissza hamisra ismét, amikor a talajra érkezett. A futás értékét ugyanilyen módszerrel változtatom. Az elkészült animátor a 17.-es ábrán látható.



17. ábra, 9.1 fejezet: A játékos animátora

9.2. 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.

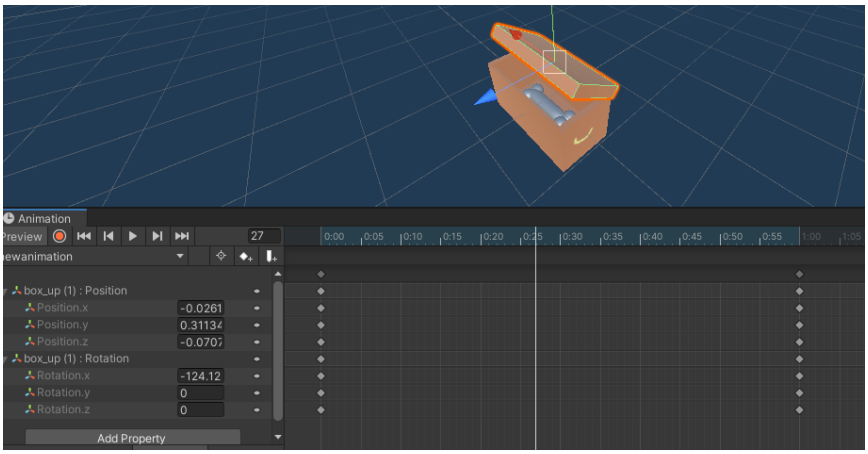
9.3. Láda animációja

A láda animátorában három animációt használtam.

– csukott állapotban
– a nyitás animációja
– nyitott állapotban

Azért volt szükségem három állásra, hogy a láda amig a játékos nem lép interakcióba vele, addig csukva maradjon és a kinyitás elvégzése után ne ugorjon vissza az animáció előtti állapotba.

A láda animációját a Unity-ben hoztam létre, mert ennek a tárgynak a meganimálásához szükséges volt mozgatni és forgatni a tárgyat. A mozgatás és forgatás animációjának beállítását a 18.-as ábra mutatja be. Nem volt szükséges a csontok használata, mivel a láda alját és a tetejét külön egységből építettem fel. Könnyebb az elhelyezkedéseket a Unity-ben átlátni, hogy hogy lesz a legmegfelelőbb a játékban. 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.



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

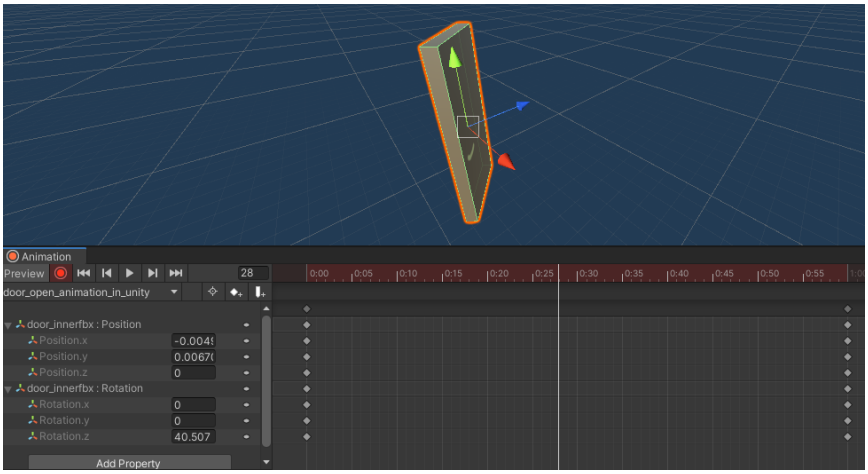
9.4. Ajtó animációja

Az ajtó animáció hasonlóan készült, mint a láda 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.



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

10. Unity prefabok

A Unity-ben a „prefabok” olyan GameObject-ek, melyeket mi hozunk létre már létező GameObjectek-ből. A prefab-ok, olyan eszközök melyeket egyszerűen be tudunk helyezni a szintérbe és az eredeti prefab tulajdonságaival fognak rendelkezni. A prefab-okat készíthetünk más prefabok-ból is, ezzel bonyolultabb egységeket létrehozva. Azokat a GameObjecte-ket, melyeket prefab-bá szeretnénk alakítani, azokat egyszerűen megfogjuk és a fájlrendszer egyik mappájába helyezük. A prefab-okat általában a Resources mappán belül egy prefab mappába kell helyezni az elkülönítés érdekében. A prefab-ok hasznossága az, hogyha változtatunk a prefab-on, akkor az össze másolata is automatikusan megváltozik. A prefab-ok az építő elemei a játékoknak. A játékban szinte minden elem, ami a szintéren letételre kerül, az egy prefab-ból származik. Ilyen például

- a kutyák
- a falak
- a padló és a mennyezet
- a csont

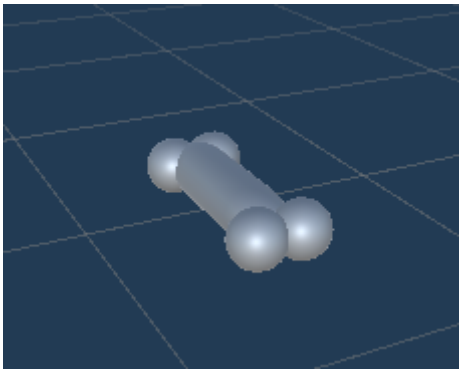
A játékban több olyan prefab-ot is használok, amiben egy másik prefab is szerepel, ilyenek

- a falak, melyeken fáklya szerepel

– a fal, amelyen van az ajtó
– a mennyezet, melyen lóg a csillár
– a láda, melyben a csont van

10.1. Csont prefab

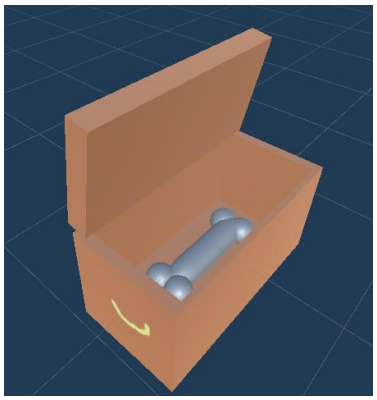
A csont prefab-ja tartalmaz négy gömböt, egy kapszula alakzatot és az azokhoz tartozó collidereket. A csont ezekből az alakzatokból épül fel, mely a 20-as ábrán látható. A csontnak van egy hasonló scriptje, mint az érmének, amely azért felel, hogy fel és le lebegjen.



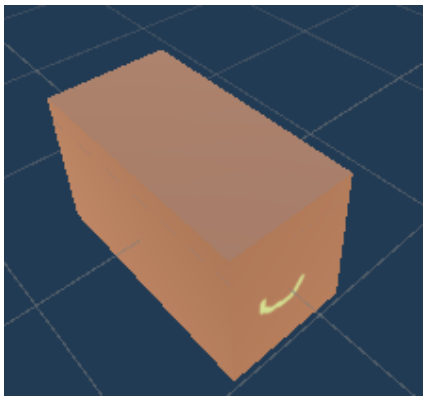
20. ábra, 10.1.fejezet: A csont prefab-ja a Unityben

10.2. Láda prefab

A láda prefab-jában szerepel a láda tetejének a modellje, a láda aljának a modellje és a csontnak a prefab-ja. 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 elvégzi a láda tetejéhez tartozó animátornak a szükséges feltételek kezelését. Az animáció lejátszódása utáni és előtti állapotot demonstrálják a 21.-es és a 22.-es ábra. Ezt a doboz prefab-ot a labirintust generáló script fogja elhelyezni a labirintusban az alapján, hogy milyen gyakori a láda a labirintusban.



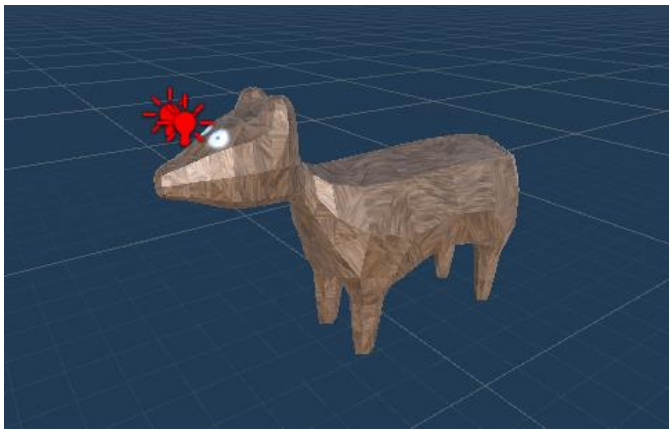
21. ábra, 10.2 fejezet: Unity-ben a láda prefab nyitott állapotban



22. ábra, 10.2 fejezet: Unity-ben a láda prefab csukott állapotban

10.3. Kutya prefab

A kutya prefab-ja tartalmazza a korábban Blender-ben 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 a labirintusban 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 prefab-ot is a labirintust generáló srcipt helyezi el a labirintusban. A játék elején a kutya mindig vár egy kis időt mielőtt elindul.

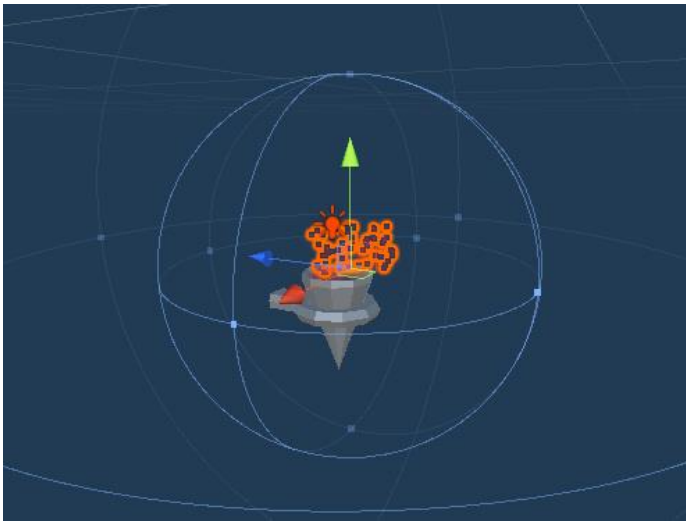


23. ábra, 10.3 fejezet: Unity-ben a kutya prefabja, melyen látszódnak a szeménél elhelyezett két fényforrás

10.4. 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 szükséges fényforrást,

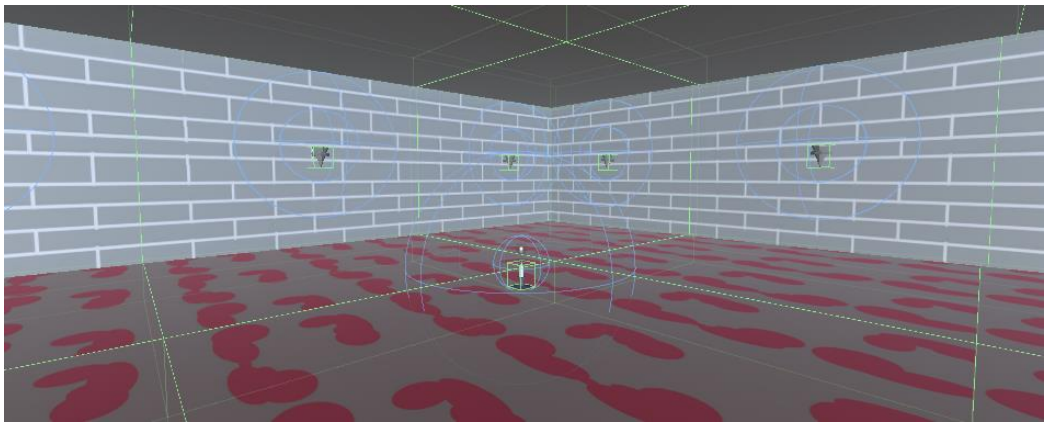
melyhez a hagyományos Point light-ot választottam. A fénynek egy narancssárgás vöröses színt állítottam be, hogy megfelelő tűz hatása legyen. A prefab-hoz még hozzáadtam egy Particle System, melyet úgy állítottam be, hogy egy széles kúp alakzatban gyorsan engedje kifele a részecskéket, melyek csak rövid ideig maradnak meg, ezzel a tűz pattogó érzetét keltve. A prefab-on 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.



24. ábra, 10.4 fejezet: Unity-ben a fáklya prefab-ja, melyen látszódik a prefab-nak az egyes elemei

10.5. Utolsó szoba prefab

Az utolsó szobára készített prefab egy háromszor hármas szoba nagyságú terület. Ahova, ha a játékos elér, akkor elhagyhatja a labirintust. Ezen prefab-ot az eddig használt prefab-ok segítségével építettem ki, melyhez még hozzáadtam 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 prefab-hoz azt a fal prefab-ot használtam, melyen a fáklya is szerepel, hogy a lehető legkivilágítottabb legyen, 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.-ös ábra mutatja meg az utolsó szoba felépítését.



25. ábra, 10.5 fejezet: Unity-ben az utolsó szobának a prefab-ja feltüntetve rajta a benne szereplő tárgyakat

Irodalom Jegyzék

[1] A Unity angol nyelvű dokumentációja: https://docs.unity3d.com/Manual/index.html

[2] A Blender angol nyelvű dokumentációja: https://docs.blender.org/manual/en/latest/

[3] Wikipédia Labirintus: https://hu.wikipedia.org/wiki/Labirintus

[4] Wikipédia Labirintus létrehozó algoritmusok: https://hu.wikipedia.org/wiki/Labirintust_l%C3%A9trehoz%C3%B3_algoritmus

[5] Wikipédia Unity: https://hu.wikipedia.org/wiki/Unity_(j%C3%A1t%C3%A9kmotor)

[6] Wikipédia Blender: https://hu.wikipedia.org/wiki/Blender_(program)

[7] Wikipédia 3D modellezés: https://hu.wikipedia.org/wiki/3D_modellez%C3%A9s

[8] Kagylókürt Labirintus cikk: http://www.kagylokurt.hu/15432/kulturtortenet/labirintus-elet-es-halal-szent-jateka.html

[9] Németh Gábor Számítógépes grafika alapjai, előadás anyag: https://www.inf.u-szeged.hu/~gnemeth/kurzusok/szggrafalap2011/handout2014/SzGrafAlap_10_megvilagitas.pdf

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: …….