Feladatkiírás

A témát kiíró oktató neve: Németh Gábor

A témát meghirdető tansuék: Képfeldolgozás és Számítógépes Grafika Tanszék

Típusa: Szakdolgozat

Hány fő jelentkezhet: 1fő, Programtervező informatikus BSc, Mérnökinformatikus BSc vagy Gazdaságinformatikus BSc szakos hallgató jelentkezhet

A feladat rövid leírása:
A játékoknak és a grafikai animációknak megjelent egy olyan stílusa, amelynél pont az alacsony felbontású poligonokkal alkotott formák adják a látvány érdekességét. A jelentkező feladata egy „low poly" stílusú terepasztal elkészítése és játékosítása. A programban a játékosnak tudnia kell újabb épületeket letenni, ezzel növelve a virtuális város lakosságát. Az épületek különböző stílusúak és funkciójuak kell, hogy legyenek. Az épületek között az utakat a program automatikusan alakítja ki, ehhez meg kell tervezni egy olyan algoritmust, amely megfelelő textúrájú útrészleteket helyez a terepasztalra. Az utakon autók automatikusan, a játékos irányítása nélkül közlekednek. A játékosítási elem fő részét képezi, hogy a város lakossága, valamint a nem lakóépületek hatással vannak az élelemellátásra, amely pont formájában mutatkozik. A játék során úgy kell újabb és újabb épületeket létrehozni, hogy ez a pontszám ki legyen egyensúlyozva. Amennyiben a játékos tétlenül várakozik, úgy a pontszám csökkeni kezd. Ha a pontszám 0-ra csökken, a játék véget ér.

Szakirodalom: főként angol nyelvű

Előismeretek: számítógépes grafikai előismeretek szükségesek

Tartalmi összefoglaló

- ***A téma megnevezése:***

„*Low poly*” terepasztal készítése

- ***A megadott feladat megfogalmazása:***

A feladat egy „low poly” stílusú terepasztal elkészítése és játékosítása. A programban a játékosnak tudnia kell újabb épületeket letenni, ezzel növelve a virtuális város lakosságát. Az épületek különböző stílusúak és funkciójuak kell, hogy legyenek. Az épületek között az utakat a program automatikusan alakítja ki, ehhez meg kell tervezni egy olyan algoritmust, amely megfelelő textúrájú útrészleteket helyez a terepasztalra. Az utakon autók automatikusan, a játékos irányítása nélkül közlekednek. A játékosítási elem fő részét képezi, hogy a város lakossága, valamint a nem lakóépületek hatással vannak az élelemellátásra, amely pont formájában mutatkozik. A játék során úgy kell újabb és újabb épületeket létrehozni, hogy ez a pontszám ki legyen egyensúlyozva. Amennyiben a játékos tétlenül várakozik, úgy a pontszám csökkeni kezd. Ha a pontszám 0-ra csökken, a játék véget ér.

- ***A megoldási mód:***

A terepasztal objektumainak tárolására asszociatív tömböket használtam. A legtöbb funkció megvalósításához tömbbejárásra és esetvizsgálatokra volt szükségem.

- ***Alkalmazott eszközök, módszerek:***

Three.Js, hash map, asszociatív tömb, blender modellezés,

- ***Elért eredmények:***

Elkészítettem egy város-építős játékot, amelyben szabadon tudunk épületeket lehelyezni, azokkal különböző életszínvonalat befolyásoló tényezőt módosítani, amiket ha nem szabályozunk megfelelően, a játékunk véget ér. A programban megvalósítottam az utak generálásának automatizálását is.

- ***Kulcsszavak:***

Three.Js, játék, szimulátor, modellezés, NodeJs

Tartalomjegyzék

(A tartalomjegyzék még pontosítva lesz, amint minden mással készen leszek)

BEVEZETÉS

A modellezés iránti érdeklődésem, annak hobbiszerű űzése adott ihletet egy olyan játék létrehozására, amelyben szabadon fel tudom használni az elkészült munkáimat. Egy ideje foglalkoztat a low poly stílusú épületek modellezése, így került a képbe egy város építő webalkalmazás elkészítése, ahol nem csak az általam készített épületek, járművek különbözőségének felfedezése adhatja a játék élményét, hanem a városunk építése során kihívásokkal is szembe kell néznünk, fenntatva ezzel a játék iránti érdeklődést. A játékom a PolyCity nevet kapta, ezzel utalva a low poly ábrázolásra, és a játék témájára is.

A cél, hogy minél kiegyensúlyozottabb várost építsünk. Különböző statisztikai tényezőknek kell megfeleltetnünk a városunkat, ami ha nem sikerül, nem tudjuk jól összehangolni a lehelyezett épületeinket, járműveinket, és nem tudjuk az időkorláton belül feloldani a kiegyensúlyozatlanság okozta problémákat, a játék véget ér.

A motivációm másik jelentős részét a gyerekkoromtól még mai napig kedvelt játékok adták. Ezek közül a legmeghatározóbbak a SimCity és a RamaCity. Ezek mellett számos online, ingyenesen és szabadon hozzáférhető, különböző, funkcionalitásokban eltérő játékokat lehet találni az interneten.

Bár az alap koncepciója a PolyCity-nek megegyezik a fentebb említett két játékéval, funkcionalitását tekintve nagyban eltér azoktól. Az egyik alapvető befolyásoló tényező, a pénz ugyanis az én verziómban nem jelenik meg, ami így egy erős korlátot old fel a játék során, azonban figyelembe vesz olyan részleteket, mint például a populáció által meghatározott munkahelyigény, valamint vásárlási igény kielégítése, amit a másik kettő játék nem tűz ki célként.

A dolgozatom következő fejezeteiben az alapfogalmak ismertetése után szeretnék szélesebb körű betekintést nyújtani a tervezés, implementáció és fejlesztés folyamatába, a kód struktúráltságába és szerkezeti felépítésébe. Részletesen tárgyalni fogom a használt algoritmusokat, főként az automatikusan generált objektumokkal és a mozgó elemekkel kapcsolatban. Végül az elkészült funkciókat mutatom be, melyek azok amiket a játék fejlesztésének kezdeténél kigondoltam, hogyan sikerült őket megterveznem és megvalósítanom. Arra is ki fogok térni a dolgozatom zárása előtt, hogy melyek azok a továbbfejlesztési lehetőségek, funkciók, amelyekkel a játékot még élvezetesebbé és esztétikusabbá lehetne tenni.

1. Alapfogalmak

1.1. Three.js

A Three.js egy böngészők közötti JavaScript-könyvtár és alkalmazásprogramozási felület (API), amely számítógépes, animált 3D grafikák létrehozására és megjelenítésére szolgál a webböngészőben, WebGL használatával.

1.2. Színtér

A színtér (Scene) reprezentálja a 3D világot, az ehhez hozzáadott objektumok jelenhetnek csak meg az eredményképen.

1.3. Objektumok térbeli elhelyezése

A beépített geometriák a világ koordináta-rendszer origójában kerülnek definiálásra. Lehetőségünk van X-, Y- és Z-tengelyek mentén elmozgatni, méretüket skálázni, és a tengelyek körül adott szögben elforgatni őket.

1.4. Alacsony poligonos ábrázolásmód

Az alacsony poligon (low poly) egy olyan sokszögháló a 3D számítógépes grafikában, amelynek viszonylag kevés sokszöge van. Az alacsony poligonok valós idejű alkalmazásokban (például játékokban) fordulnak elő, ellentétben az animációs filmekben szereplő magas polimetrikus hálókkal. A low poly kifejezést technikai és leíró értelemben is használják, a sokszögek száma egy hálóban fontos tényező a teljesítmény optimalizálása szempontjából, de nemkívánatos megjelenést kölcsönözhet a kapott grafikának. A szakdolgozatom során mint stílust emlegetem, mivel ennek az ábrázolásmódnak köszönhetően a magas polimetrikus hálók ábrázolásával szemben egy teljesen más vizuális hatást tudunk elérni.

1.5. Ambiens fény, vagy szórt, környezeti fény

Nem tekintjük igazi fényforrásnak, mert nincs forrása, sem iránya, csak színt és intenzitást rendelhetünk hozzá. A tárgyak színe a fény színéből és a tárgyak anyagának szín jellemzőjéből számolódik. Önállóan ritkán használjuk, de más fényforrásokkal kombinálva egy globális színhatást tud biztosítani a színtérben.

1.6. Pontfény

A tér egy megadott pontjából a tér minden irányába indulnak fénysugarak. A fényhez gyengülési tényező rendelhető, vagyis a forrástól távolabbi tárgyakat már gyengébben világít(hat)ja meg.

1.7. Kamera

A kamera (Camera) paraméterei határozzák meg, hogy a 3D világ hogyan kerüljön levetítésre a 2D eredményképre. A Three.js kétféle kamerát biztosít: párhuzamos és perspektív vetítést végrehajtót. Ezek közül a perspektív az, amely az emberi látáshoz közelebb van, a párhuzamos inkább a mérnöki alkalmazásokban fontosabb. A kamerának megadhatjuk a térbeli pozícióját, a vetítés irányát (hová néz), valamint a kamera felfelé mutató irányát, ugyanis a vetítési tengely körül is el lehet forgatni.

2. Szerkezeti felépítés

2.1. Általánosan a kód struktúrájáról

A dolgozatom elkészítése során Node.js-t, azon belül pedig three.js-t használtam. A Node.js egy olyan környezet, amely lehetővé teszi a JavaScript futtatását a szerveroldalon. Ez azt jelenti, hogy a kiszolgáló gépen JavaScript kód fut, amely kérésekre válaszolhat, kommunikál más szolgáltatásokkal, hozzáférhet az adatbázishoz. Az én esetemben adatbázis nincs, minden adatot a JavaScript fájlokban, illetve assetekben (Blender modellek) tárolok el. Ezenkívül a Node.js lehetővé teszi a JSON fájlok kezelését is, amelyek egy könnyen olvasható, szöveges adatfájl-formátum. JSON formátumot használtam a városom objektumainak leírására, amelyeket majd a Three.js használ a megjelenítéshez.

2.2. Html oldalak

Az oldal 2 html fájlból tevődik össze. Az *index.html* oldal már hangulatában is előkészti a játékota, az abban használt színekkel és képekkel. Emellett itt találhatók a logo, a játékleírások, valamint egy gomb, amely a *game.html* oldalra navigál minket. A szükséges modellek és textúrák betöltési ideje alatt egy betöltő képernyő jelenik meg az oldalon. Amint ez a folyamat lezajlott, megjelenik előttünk a játéktér, amely bal oldalán megtalálható egy információs és eszköz panel. Itt a lehelyezhető épületeket és járműveket, statisztikai adataikat, valamint a modellek előnézeti képeit láthatjuk. Itt tudjuk az adott épületre való kattintással a kiválaszott elemet a játéktérre helyezni. A panel felső részén megtalálhatóak a városra vonatkozó életszínvonal paraméterek. Ezen számadatok helytelen alakulása esetén egy számláló doboz jelenik meg az oldal tetején középen. Ha a játékot elvesztettük, ugyan ezen az oldalon jelenik meg egy eddig rejtett doboz két gombbal, amely a játékteret eltakarja, így az tovább már nem használható, a játékot újra kell kezdeni, vagy vissza lehet menni a főoldalra.

2.3. JavaScript fájlok

A kód 5 JavaScript fájlba van kiszervezve, ezek a *functions.js, globals.js, listeners.js, loader.js,* valamint a *script.js*. Mivel rengeteg különböző funkciót kellett implementálnom, a könnyebb átláthatóság és a jól struktúráltság érdekében külön egységekbe szerveztem a kódot. A különböző fájlokban implementált kódrészleteket, függvényeket, változókat *export default* segítségével tettem elérhetővé a többi fájl számára, ahol az importokon keresztül ezek használhatóvá is váltak.

2.3.1. Globális változók, konstansok

A *globals.js* fájlba szerveztem ki a globális változókat, amelyeket minden további fájlba beimportáltam, használatuk az egész programra kiterjed. Más javascript fájlok importjait nem tartalmazza, hiszen itt hozom létre az alap konstansokat, változókat, ezen kívül semmi más nem történik ebben a fájlban. Egy globals nevű objektumban tároltam le azokat az adatokat, amelyeknek nincs szükségük egymás használatára. Ilyenek voltak a képernyő méret adatai, az objektumok lehelyezéséhez szükséges navigáció alapadatai, az utak, valamint az épületek koordinátáinak eltárolására használt map-ek, az objektumok kategorizálására szolgáló számadatok, a város statisztikai adatai, maga a scene és annak paraméterei, ég és föld objektumok, navigációt segítő objektum, autók eltárolására szolgáló tömb, egy, az autók irányának meghatározásában szerepet játszó tömb, irány adatokkal. Azok a globális paraméterek, amelyek valamely másik globális konstans vagy változó értékét használják fel, a globals objektumon kívül helyezkednek el. Ezek a kamera, mozgatás és renderelés inicializálásai.

2.3.2. Függvények

A *function.js* szolgál a programban használt fő függvények öszegyűjtésére. Ez a programkód leghosszabb része, a játék funkcionalitásáért leginkább felelős részleteket találhatjuk meg ebben a fájlban. Itt valósítom meg a pálya növekedéséért felelős függvényt, az útgenerálást, utak kirajzolását, azok újraépítéséhez szükséges függvényeket, valamint az annak ellenőrzésére szolgáló kódrészletet, hogy az éppen vizsgált koordinátán van-e objektum, és ha igen, az út-e. Itt kerül implementálásra az autók mozgatásáért felelős függvény, valamint a játék fő mechanizmusát képző statisztikai adatok határértékének vizsgálata is. Egyedül a script.js fájlban importálom, ahol csupán az autók mozgatásáért felelős függvényt hívjuk meg.

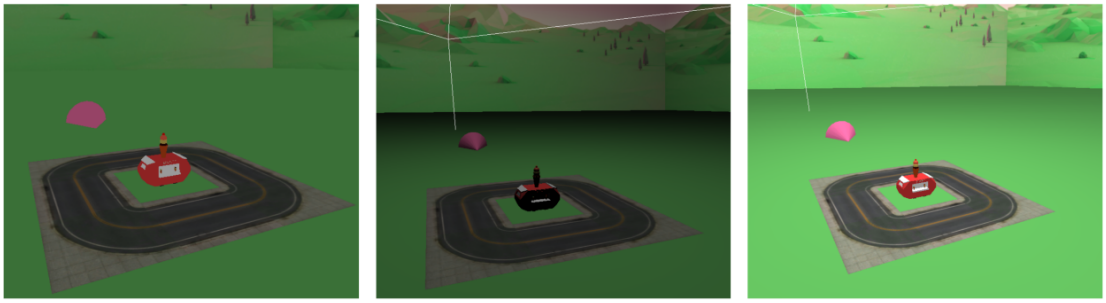
2.3.3. Betöltők

Az utak textúráinak, valamint a modelleknek a betöltése a *loaders.js* fájlban történik. Ezt egy nagy aszinkron függvényben valósítom meg, amelyet a megfelelő helyen meghívva gondoskodok arról, hogy a játék ne indulhasson el a töltési folyamatok befejezése előtt. Itt állítom be a játéktérben elhelyezendő objektumok méreteit is, mivel azok modell szerinti méretei jelentősen eltértek egymástól, így nem alkalmazhattam mindgyik modell esetén ugyanazt a méretezést. Mindegyik esetén meg kellett néznem, mik azok az arányok, amivel pontosan belefférnek a szomszédos utak által alkotott keretbe. Ezt a betöltésre szolgáló függvényt egyedül a script.js fájlban importálom, ahol azt egyszer hívom meg az init függvény előtt, így még minden más előtt le fog futni.

2.3.4. Fő script fájl

A script.js fájl gyakorlatilag egy fő JavaScript fájlként funkcionál, így semelyik másik fájlban nem kerül importálásra. Itt valósítom meg azt az aszinkron függvényt, amely felelős azért, hogy a játék csak akkor kezdődjön el, amikor a szükséges modellek, textúrák már betöltődtek. Ezt a betöltési folyamatot egy betöltőképernyővel rejtem el a felhasználó elől. Emellett a globális változóknál létrehozott objektumok, valamint a játéktér falainak részletes paramétermegadása is itt történik.

Ebben a fáljban hozom létre pálya megvilágításához szükséges fényeket is. Ha a játékteret nem világítanám meg, sem a lehelyezett objektumok, sem maga a pálya nem látszódna, hiszen a falak, a föld és az ég is objektumok, amik megvilágítás nélkül láthatatlanok. Kétféle megvilágításra volt szükségem a kívánt eredmény elérésének érdekében. Szükség volt elsősorban egy ambiens fényre, amely egy úgymond alapfényként szolgál a játékban. Ez egy 0.6 erősségű fehér fény, amely már elegendő lenne a láthatatlanság kiküszöbölésére, azonban ezzel igencsak átlagos, torz képet kapunk csupán az épületeinkről, nem emeli ki azok részleteit, ami a játékélményt véleményem szerint nagyban rontotta. Ennek megoldására szolgál a lehelyezett pontfény, amely a pálya közepéhez lett igazítva, azonban az egész pályára kiterjed annak hatása. Ez a két fajta fényforrás teszi lehetővé tehát a modellek tökéletes megjelenítését. A különbségeket az alábbi képsorozaton láthatjuk, az elsőn csak ambiens fényt, a másodikon csak pontfényt, a harmadikon pedig mind a kettőnek a használatát.



2.1 ábra: Ambiens fény, pontfény és a kettő kombinációja.

Az idő követésére szükséges konstans és a hozzá tartozó metódus is itt található, itt kötöm be az eseményfigyelőben implementált ablakátméretező függvényt, inicializálom a képernyő bal oldalsó, statisztikai adatokat megjelenítő sávjában feltüntetett adatokat, azaz ebben a fájlban kötöm össze a html kódot és a JavaScript-ben megírt és a képernyőre kiírandó változókat. A fájl végén pedig meghívom az autók mozgatásáért felelős függvényt.

2.3.5. Eseményfigyelők

Az eseményfigyelőket a listener.js fájlban valósítom meg. Ebben a fájlban található az ablak átméretezéséért felelős kódrészlet, valamint az itt létrehozott függvények felelősek az összes felhasználó általi interakcióért, azok feldolgozásáért és a játékban történö felhasználásáért, többi fájlnak való továbbításáért. Itt valósítom meg az autók hozzáadása függvényt, ahol ellenőrzőm, hogy a kiválasztott hely út-e, és csak akkor hajtom végre az objektum lehelyezését, ha ez igazra értékelődött ki.

Az objektumok lehelyezése a képernyő bal oldalán található panel segítségével lehetséges. A kiválasztott objektum előnézeti képére kattintva a navigálást segítő objektum helyén megjelenik a modell. Ez az egér gombnyomásának html objektum egyedi azonosítójához kötött eseményfigyelőjével került megvalósításra. Minden azonosítóhoz, így minden előnézeti képhez tartozik egy saját függvény, amely kattintáskor kerül megnyitásra. Ez a függvény minden esetben leellenőrzi, hogy a lehelyezendő épület kiválasztott helye nem foglalt-e még, és amennyiben ez igaz, a betöltött objektumok közül létrehozunk egy másolatot a válaszott épületről, beállítjuk annak helyadatait a navigálás segítő pozíciójával megegyezőre. Ez után a forgatást is elvégezzük egy globális változó segítségével, majd az objektumokat tároló map-ben beállítjuk a koordinátákat, valamint a hozzájuk rendelt számértéket, amely a lehelyezett objektum típusát adja meg. Ez a szám fog nekünk segítségül szolgálni az utak kiépítésében, valamint a járművek lerakásában is. Amint ez kész, hozzáadom a játéktérhez az elkészült másolatot. Azért használok másolatot, mert a modelleket csak egyszer töltődnek be, a játék elején a betöltő fázisban, annak érdekében, hogy ezek a folyamatok ne lassítsák be a játékot, ne rontsák a felhasználói élményt, ne használjanak fel túl sok erőforrást. Végül meghívom a játéktér növelő függvényt, valamint az útgeneráló függvényt a lehelyezett objektum x és z koordinátájával, majd a statisztikai adatok határértékének vizsgálatára szolgáló függvényt is.

A járművek lehelyezése esetén majdnem mindenben különbözik a meghívott függvény, az épületekétől. Az ellenőrző feltétel itt azt vizsgálja, hogy a koordináták által meghatározott mezőn út helyezkedik-e el. Ha igen, akkor másolatot készítünk ebben az esetben is a már betöltött objektumról, valamint a pozíciók előzőhöz hasonló beállítása után hozzá is adjuk őket a színtérhez. Ez után megadunk nekik egy véletlenszerű irányt. Ezt úgy tudjuk megtenni, hogy a globális változókban megírt 4 elemet tartalmazó tömbből kiválasztjuk véletlenszerűen az egyik elemet. Ehhez a Math.floor és a Math.random beépített függvényeket használtam. A random függény eredményét be kellett szoroznom 4-el, majd vennem kellett az alsó kerekítését a floor segítségével, hogy 0 és 3 közötti egész számokat

kapjak, hiszen ezek adják majd meg a tömböm koordinátáit. Ekkor egy feltételes szerkezetben megadom, hogy az adott eredményekre mennyivel és milyen irányba forgassa el a járművet. Végül az objektumot annak irányával hozzáadom egy autókat tartalmazó tömbhöz, valamint az autók számát megnövelem egyel. Majd meghívom a statisztikai adatok határértékeinek ellenőrzésére szolgáló függvényt is.

Az egérrel való kattintás figyelése mellett szükségem volt még a space billentyű figyelésére, amely a navigációt segítő objektumot forgatta el minden egyes gomb lenyomása után 90 fokkal. A le, fel, jobbra, balra nyílbillentyűk lenyomása esetén beállítottaam, hogy a helyválasztó a megadott irányokba mozduljon el, ezzel annak koordinátáit is módosítva. A programban biztosítva van, hogy ezek az értékek ne vehessenek fel nagyobbat a pálya mérének felénél sem negatív, sem pozitív irányba, így azt is szabályzom, hogy a nevigáló segíőm ne mehessen le a pályáról, aminek következtében tehát a pályán kívüli objektumlehelyezés sem lehetséges.

3. Tervezési folyamatok, előkészületek

A játék megtervezése kezdetekben magának a játék stílusának, a modellek kinézetének és színvilágának összehangolása, a látvány elképzelése, vázlatok, modelltervek készítése volt. Ezt a folyamatot követően próbáltam felhasználói szempontból megközelíteni a feladatot, mi lenne az, ami a legtöbb játékélményt nyújtaná, mivel lehetne változatosabbá, nagyobb kihívássá tenni a játékot. Ezután a technikai részletekben merültem el, milyen algoritmusok használatával, milyen adatszerkezetekkel fogom tudni megvalósítani az egyes részleteket, mik azok, amikre az ismeretem még nem terjedt ki és további kutatómunkát, önfejlesztést igényelnek. Végül a fejlesztési folyamat után jött még egy kisebb tervezési folyamat, mik azok a funkciók, amiket még meg tudok csinálni az adott időn belül, mik azok, amiket még mindenképp szeretnék megvalósítani a saját elégedettségem elérése érdekében.

3.1. Stíluselemek, modellek, látványelemek megtervezése

A lowpoly modellezés már egy ideje nagyon a szívemhez nőtt, így a fejlesztés során nagy örömöt okozott, hogy egy nagyobb projektemben felhasználhatom az elkészített modelljeimet. A modellek stílusa mellett a színük is meghatározott volt. Az elkészítés során egy 40 árnyalatból álló színskálát használtam, így nagyobb összhangot teremtve a modelljeim között. Mindegyiket gondos előkészületek, tervezési munkálatok után készítettem el, hasonló munkákból vettem inspirációt, szakmabeliek trükkjeit tanultam el és alkalmaztam a saját munkálataim során. Először tehát a modellek hangulatát kezdtem el megtervezni és csak ez után jött magának a játéknak a stílusbeli tervezési folyamata.

A színeket és a fényviszonyokat próbáltam az épületekhez igazítani, világos és kellemes színeket használtam mindig. A hangulatteremtéssel is ez volt a célom, szerettem volna, ha egy kellemes, vidám játék kerekedik ki a fejlesztésem végére. Sok város-építős játéknál láttam a realista ábrázolást, a szürke épületekben, végtelen zsúfolt úthálózatokban, ami engem sokszor eltántorított a játéktól, nem szívesen ültem le olyannal játszani, aminek az ábrázolása nem illeszkedett az elvárásaimhoz, ezért követtem egy ezektől eltérő, inkább mesevilághoz közelítő ábrázolásmódot.

A főoldalon és a játék hátthereként is megjelenő képet az inteneten találtam, forrását megjelöltem az irodalomjegyzékben. Ennek a képnek azonban egy átszerkesztett változatát használtam, mert nem illeszkedett eléggé ahhoz a színvilághoz, amit terveztem, így módosítottam ezek alapján az eredeti változatot.

3.2. Játékmechanika megtervezése

Elsődleges célom az épületek lehelyezéséből adódó játékélmény megteremtése volt. Nem sok gondolkodás kellett, hogy ez az épületlehelyezgetés önmagában kicsit unalmas ötletté váljon, így jutottam el az utak rendszerének kigondolásáig. Fejlesztés előtt felmerült bennem az ötlet, hogy a játékos helyezze-e le az utakat az épület köré, vagy a játék generálja azokat. Az automatikus megoldás mellett döntöttem, mivel számos más internetes játékban az előbbi megoldást alkalmazták, én pedig amiben csak tudtam, igyekeztem a meglévőktől különbözőt alkotni, így ebben is a bár nehezebb, de különbséget képező megoldást választottam. Emellett nem tetszett annak az ötlete sem, hogy egy rendszerezetlen, csak épületekből álló városképet is ki lehetett volna hozni a játékbol, ha az utak lehelyezése nincsen szabályozva, így az esztétikai elvárásaimat is támogatva a generált utak megoldása mellett maradtam.

Ez után jött az autók lehelyezésének ötlete, ami mellett már a kezdetekben is megjelent azok automatikus mozgatásának ötlete is, hiszen valamilyen szintű animációt is szerettem volna a játékban bemutatni. Gondoltam arra is, hogy ez az animáció megjelenjen-e a modelljeimben is, azonban annyira még nem vagyok fejlett a modellezésben, hogy esztétikus animációkat készítsek velük, a továbbiakban fejlesztési lehetőségnek viszont remek ötletnek tartanám.

Volt egy pont a fejlesztés közben, amikor úgy gondoltam, hogy az utak összekötése nem szükséges, olyan értelemben, hogy távolabbi városrészeket nem kötök össze. Ez azonban hamar kideült, hogy jelentősen lecsökkentené mind látványban mind játékmechanikailag is az elvárt szintet.

A játéktér átméretezésének lehetőségét fontos dolognak tarttottam, nem akartam a pálya meghatározott méretével lekorlátozni a lehetőségeket, viszont kezdetben nem szerettem volna túl nagy pályát biztosítani, mivel akkor a háttér, valamint az ég nem látszott volna eléggé, valamint a kevés lehelyezett objektum miatt unalmasan nézett volna ki csak a zöld mező látványa. Annak biztosítása is fontos volt, hogy a kamera mozgatható legyen, mivel ha egy nagyobb várost építünk meg, szerettem volna, ha arra ráközelítve végig tudunk pásztázni a látványképen, meg tudjuk figyelni közelebbről a távolabbi részleteket is.

Az már a tervezési folyamatok elején letisztázódott benne, hogy a forgalomirányítás lesz az a tényező, amit nem szeretnék megvalósítani. A modellek méretét jelentősen le kellett volna csökkentenem ahoz, hogy elférjenek kényelmesen egymás mellett, emellett az animáció várakoztatására is szükség lett volna a járművek fordulásánál, azok kereszteződésben való elhaladásánál, a pozíciók kiszámításánál, megtett út alapján a fordulás helyes időpontban való

végrehajtásánál, így az ebbe fektetett energia helyett inkább más funkcionalitások megvalósítására koncentráltam.

Az alapkoncepció ellenére a játék fő mechanizmusává a kezdetben nem tervezett funkció vált, a statisztikai adatok figyelése, azok helyes alakulásának elősegítése, a játékos fejében kialakuló tervezési folyamat, a számolgatás, mit építsen le annak érdekében, hogy a játéka ne érjen véget. Kezdetekben csak a tervezés és az esztétikus látvány megteremtése adta játékélmény nyújtása volt az elvárásom, ám ez idővel kevésnek bizonyult. Ekkor jöttek be a különböző életszínvonalat befolyásoló adatok, azok épületekkel és játművekkel való alakításának ötlete. Ez eleinte még csak a munkahelyigény, a vásárlási igény, valamint a szórakozás lehetőségének a szintje volt. Emellett a populáció is feltüntetésre került, amely csupán tájékoztató jelleggel bír, ezzel további feladata nincs a játékosnak. Később úgy éreztem ez az adat igencsak kevés, gondolkoztam új tényezők beépítésén. Ennek eredményeképp került beépítésre a szennyezettség, valamint a közellátás, gondolok itt az általános szolgáltatásokra, mint a rendőrség, kórház.

Eleinte ezeknek csak a megjelnítését terveztem, úgy gondoltam elég, ha a felhasználó saját megítélése alapján eldönti, hogy olyan várost épít, amelyben fontos egyes tényezők figyelembe vétele, a többire pedig nem koncentrál, emellett meg szerettem volna adni arra a lehetőséget, hogy a játékos ne ezen adatok szerint, hanem a saját esztétikai igénye alapján, logikai felépítettség nélkül alkossa meg a városát, azonban végül arra jutottam, hogy megszabok ezekre az értékekre egy korlátot, ami szerint a játékot nehezítem, így már célt is tűzök ki a játékosok elé. Ezzel véleményem szerint az újrajátszhatósága növekedett a játékomnak, már a kihívás is motiváviós tényezőként hat a játékosra. Így tehát ha a megengedett határértékeket átléptük, egy rövid időt kapunk arra, amelynek számlálója meg is jelenik, hogy a problémát feloldjuk, ez után azonban a város boldogtalanná válik, nem szeretnék a lakosok, hogy a játékos irányítsa az ő városukat, így kirúgják a város vezetéséből, a játék véget ér.

3.3. Technikai tervezés

A tervezés utolsó fázisában került sor a program felépítésének, az adatstruktúrának, algoritmusoknak a kidolgozására. Itt először is a pálya egyes területegységeinek eltárolási módjára volt szükségem, ami könnyen látszott, hogy egy asszociatív tömb lesz. Ebből kettőt is készítettem, egyet az utaknak, egyet pedig minden más, a játékos által lehelyezett objektumnak. El kellett tárolnom az adott pozíciók koordinátáit, amelyet egy sztringben valósítottam meg, mégpedig úgy, hogy az x és a z koordinátát egy alulvonással összekötöttem, így később könnyebb volt a map kulcsai között a keresés. Ezek után mindegyik kulcshoz kellett rendelnem egy számot, amely a lehelyezett, vagy generált objektum kategorizálására szolgáló szám volt. Ez a szám három féle kategóriát jelentett, az utak, épületek, járművek külön számtartományba kerültek. Az utak ezen tartományon belül a különböző irányú, azaz vízszintes egyenes, függőleges egyenes utak, négy féle kanyar, három elágazású kereszteződések, valamint azok irányai, négy elágazású kereszteződések számjelzését kapták.

Amint tisztázódott, miképpen tárolom el a játék objektumait, meg kellett terveznem a szükséges algoritmusokat, amelyek megmondják, melyik mezőkre és milyen irányokba kell az utakat lehelyezni. Ez után azok összekötésére és az autók mozgatására szolgáló függvények kigondolására fektettem a hangsúlyt. Majd a falak és pálya növekedésének mértéke, azok lehelyezés függvényében való változása került megtervezésre.

Legvégül a statisztikai adatok összehangolása, azok bekötése a játékba maradt feladatként. Itt szükség volt arra, hogy a lehelyezhető épületek és járművek közelítve a valós életben betöltött funkciókhoz, hozzájuk arányosan változtassák a város paramétereit, ki kellett tapasztalnom, hogy melyek azok a tartományok, hatérértékek, amelyeket a kiegyensúlyozatlanság után következő egy perces időkorlát alatt még vissza lehet állítani.

4. Implementáció

4.1. Adatstruktúrák

A játék alapjául szolgál egy asszociatív tömb, amelyre szinte az összes funkcionalitás megalkotásakor, valamint minden használt algoritmusnál szükségem volt. Ebben tárolok el minden művelet eredményeként létrejött pályaváltozást, ez tartalmaz minden játékban lehelyezett elemet. A generált utakat a pályán lévő, a játékos által lerakott többi objektumtól elkülönülő tömbben is eltárolom, így amikor csak az utak ellenőrzésére kerül a sor, nem kell minden egyes objektumon végigmennünk, elég csak az utak tömbjét figyelni. Ezt ugyan csak egy asszociatív tömbbel oldottam meg, szintén helyadatok és úttípus eltárolásával. Ezeken kívül egy nagy összefoglaló, globals nevű objektumban tárolok el minden szükséges változót, kulcs-érték párokként, ahol az érték lehet egy egész, lehet egy map, valamint lehet szintén egy objektum, Itt valósítom meg a programhoz szükséges inicializálásokat is, mint például a kamera, vagy a renderer.

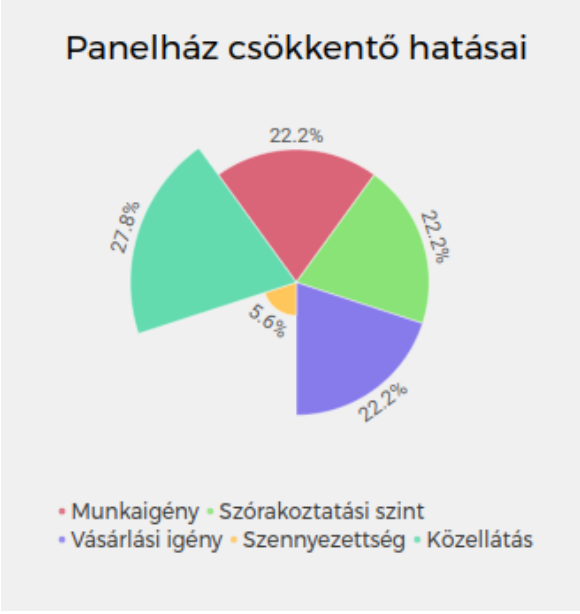
4.2. Statisztikai adatok alakulása és azok bekötése a játékba

A játék során az épületek és a járművek különbözőképpen alakítják a város statisztikáit. Építkezés során 5 paramétert kell a lehető legnagyobb egyensúlyba hozni, amelybe a populáció nem tartozik bele, ez csupán egy érdekesség a városlakóinak számáról. Minden lehelyezett épület és jármű rendelkezik saját adatokkal, amelyek megszabják, hogy melyik mennyivel növel, vagy csökkent bizonyos tényezőket. Ezek a számok az objektumok lehelyezésekor automatikusan levonódnak, vagy hozzáadódnak a város paramétereihez, amelyet a játékos azonnal látni is fog a bal oldali panel tetején. Lakóépületek lehelyezésével például a populáció növekszik, a munkahelyigény, vásárlási igény megnő, a szórakoztatási szint viszont lecsökken. Ezek kiegyensúlyozására szükséges boltokat, szórakozóhelyeket és munkahelyeket biztosító épületeket lehelyeznünk, amelyek természetesen ezen paraméterek szabályozásán túl majdnem minden más tényezőt is befolyásolnak, így nehezítve a játékot. A számokat próbáltam úgy alakítani, hogy ne legyen túl nehéz a játék.

A munkahelyigény kivételével minden tényezőt úgy alakítottam, hogy pontosan ugyan annyival lehessen növelni, mint amennyivel lehet csökkenteni az értéket, abban az esetben, ha minden, az adott tényezőt befolyásoló épületet lehelyezünk. A következő ábrán láthatóak a szórakoztatási szintet befolyásoló épületek, valamint járművek hatásai. Amennyiben összeadjuk a befolyásolás mértékeit, jól látszik, hogy az előző állításom itt is teljesül.

4.1 ábra: Szórakoztatási szintet befolyásoló épületek, járművek hatásai

Csupán kettő olyan épület létezik a játékban, amely minden tényezőt befolyásol, ezek a lakóépületek. Mivel egyedül a panelház és a családi ház ad populációt a játékunkhoz, így ezek az épületek logikusan a városba költöztetett emberek igényeit, általuk kifejtett hatást mutatják. A panelház statisztikai adatokra gyakorolt befolyását a következő ábra mutatja. Mivel a populáció növekedését kivéve, az összes tényezőt csökkenti, így csak a negatív hatásokat tüntettem fel az ábrán.



4.2.ábra: Panelház csökkentő hatásai

Amennyiben valamely paraméter értéke kilép a megengedett tartományból, elindul egy számláló. Ez az idő alatt kell a játékosnak helyrehoznia az okozott hibát, olyan objektumokat kell lehelyeznie, amelyek visszalendítik az értékeket az elfogadható szintre. Ez az időkorlát az adatok sokszínű változási lehetősége miatt, a rengeteg befolyásoló tényező miatt igen nehézkes, át kell gondolnunk mely épületek húzhatnak ki minket a bajból, mert könnyen átkerülhetünk egy olyan paraméterállásba, amit a ketyegő idő alatt már nem fogunk tudni megfelelően kezelni. Amennyiben a játékosnak nem sikerül a megadott idő alatt a számadatokat kiegyensúlyoznia, a játéktér eltűnik, a játékunk véget ér. Ez után a figyelmeztető szöveg alatt megjelenik egy gomb amivel a legelejéről kezdhetjük az építkezést, vagy akár visszamehetünk a főmenübe is.

4.3. Assetek betöltése

A script.js fájl elején, az init függvény előtt elindítok egy aszinkron függvényt, amely arra szolgál, hogy a modellek és az utakhoz, valamint a játék falaihoz szükséges textúrák a játék elindulása előtt betöltődjenek. Erre azért van szükség, mert előfordulhatna olyan helyzet, hogy még mielőtt a modellek ténylegesen betöltődtek volna, lehelyezésre kerülnek a játékos által, azonban mivel nem töltöttek még be, azok nem fognak megjelenni a játéktéren.

Ezeket a betöltéseket a function.js modellbetötő függvényében hajtom végre. Ugyan itt inicializálom a fal objektumokat is, hiszen ezek a föld és az ég objektumhoz képest textúrával rendelkeznek, amiknek ugyancsak szükséges a betöltéseük.

4.4. Függvények

A program megírása számos saját függvényre volt szükségem, amelyek általában valamilyen koordináta, vagy az azon elhelyezkedő objektum típusának vizsgálatát végezte. Ezekben a függvényekben legtöbbször elágazásos vezérlési szerkezetet használtam, de előfordult az esetkiválasztásos vezérlési szerkezet használata is. Majdnem minden esetben egy tömb bejárással kezdődött a függvényem, amelyet forEach vagy szimpla for ciklussal valósítottam meg.

4.4.1. Játéktér átméretezése

A pálya a játék kezdetekor egy 100 × 100-as méretű mátrix, mely az építkezések során automatikusan növekszik. Ha a pálya széléhez közeledve építkezünk, a játéktér annyi egységgel fog nőni, hogy a lehelyezett objektum és a határvonal között 10 egységnyi terület maradjon. Azaz ha teljesen a szélére helyezzük le az épületünket, 10 egységgel fog megnövekedni a pálya mérete. Emellett szükséges a többi fal méretének és azok pozícióinak megváltoztatása. Amennyiben a föld objektum megnő, ugyan annyi egységgel kell a falak, valamint az ég objektum méretét is növelni.

```
if (Math.abs(x) - -10 > globals.land.scale.x / 2) {
   let old_size = globals.wall_west.scale.x;

   globals.land.scale.y += Math.ceil(
     Math.abs(x) - -10 - globals.land.scale.x / 2
   );
   globals.land.scale.x += Math.ceil(
     Math.abs(x) - -10 - globals.land.scale.x / 2
   );
  //...
}
```

4.3 kódrészlet: x koordináta értékének határátlépése esetén a talaj méretének növelése

Emellett a méretváltozások okozta pozíció elcsúszásokkal is számolni kell. Annyival kell eltolni a falakat a megfelelő irányba, valamint annyival kell megemelni az eget, amennyi egységgel az egyes elemek megnőttek.

Mivel a kamera pályáról való kimozgatását is a játéktér méretével állítjuk be, így a pályanövekedés esetén a kamera maximum távolságát is meg kell növelnünk a változás értékével.

```
globals.orbit.maxDistance += Math.ceil(Math.abs(x) - -10 - globals.land.scale.x / 2) / 2;
```

4.3 kódrészlet: Kamera pályáról való kinavigálásának megakadályozása

Összességében tehát ez a funkció biztosítja a lehetőséget a városunk szinte végtelenségig fejlesztésére. Az ehhez szükséges függvényt a function.js fájlban valósítottam meg.

4.4.2. Út objektum vizsgálata

Számos kódrészletben szükségem volt annak ellenőrzésére, hogy az adott esetben vizsgált koorináták helyén út található-e, így ezt kiszerveztem egy külön, 1 soros függvénnyé, aminek a neve az *isRoad()*. Mivel minden koordinátapár által meghatározott helynek van egy hozzárendelt számadata, így elég volt csak megvizsgálnom, hogy a paraméterben kapott érték az utak értéktartományába esik-e. Ennek párja a *notRoad()* függvény, amely az előzőekhez képest a tartományon kívüliséget ellenőrzi, valamint, hogy a kapott érték *undefined*-e.

4.4.3. Utak kirajzolása

Végig megyünk az objektumok szótárának kulcs-érték párjain, és minden esetben szétválaszjuk a kulcsot egy *x* és egy *z* koordinátára. Erre azért van szükség, mivel az asszociatív tömbben a kulcsokat a két helyadat sztringgé való összekapcsolásaként *(x_z)* tároltam el. Ez után létrehozok minden lehetséges útfajtának egy anyag jellemzőt, azaz materialt. Meghívom az utak vizsgálására megírt függvényt a jelenleg vizsgált kulcs-érték párom értékével, és amennyiben ez igazra értékelődik ki, tehát utat kaptam, és ez az út még nincs benne az utak asszociatív tömbjében, valamint az utam típusa nem egyezik meg a végső úttal, azaz a négy elágazású kereszteződéssel, amelyet már nem lehetne módosítani, tovább vizsgálom az értéket. Ezúttal megnézem, hogy a kapott érték melyik útfajtával egyezik meg, és ez által létrehozok minden esetben egy 1×1 méretű négyzetlapot, amelynek a számnak megfelelő textúrát adom. Ez után annak függvényében, hogy az út milyen irányban áll, elforgatom a lehelyezett lap objektumomat a megfelelő szögben, beállítom a pozícióját, majd hozzáadom a színtérhez és eltárolom a helyadatait, valamit a típusszámát az utak asszociatív tömbjében, majd kilépek az eset ágából, ezzel leállítva a további vizsgálatokat.

```
case globals.groundObject.road.road_down_right:
        const road2 = new THREE.Mesh(
          new THREE.PlaneBufferGeometry(1, 1),
          cornerMaterial
        );
        road2.rotation.x = -Math.PI * 0.5;
        road2.position.set(xKey, globals.land.position.y + 0.01, zKey);
        globals.scene.add(road2);
        globals.roads.set(key, value);
        break;
```

4.3 kódrészlet: Jobbra le kanyarodó út kirajzolása

Ezen függvény eredményeképp olyan utakat kaptam, amelyek már a helyes irányba állnak, és kereszteződések lehetősége esetén azok összekapcsolásra kerültek.

4.4.4. Utak újraépítése

Ismét végig megyünk az objektumaink tömbjének elemein, szétválaszjuk a kulcsokat koordinátákra, de csak abban az esetben, ha utat kaptunk. Ez után egy feltételes szerkezetben megvizsgáljuk, hogy a jelenleg vizsgált elem felett, alatt, és mellett lévő két elem út, vagy nem út objektum-e. Például legelőször megnézzük, hogy ha a vizsgált elemtől mind a négy irányban út helyezkedik el, akkor a kulcs értékét átírjuk a négy irányú kereszteződés értékére, hogy később majd az út kirajzolásánál az ennek megfelelő textúrát kapja.

```
if (
    isRoad(globals.map.get(`${xKey - 1}_${zKey}`)) &&
    isRoad(globals.map.get(`${xKey - -1}_${zKey}`)) &&
    isRoad(globals.map.get(`${xKey}_${zKey - 1}`)) &&
    isRoad(globals.map.get(`${xKey}_${zKey - -1}`))
) {
    globals.map.set(
        `${xKey}_${zKey}`,
        globals.groundObject.road.four_crossing
    );
}
```

4.1 kódrészlet: Négy elágazású kereszteződés vizsgálata

Másik esetet nézve például, ha az út felett és tőle balra helyezkedik csak el út, jobbra és alatta pedig nincs, akkor az egy szimpla, balról felfele irányuló kanyar lesz. így járunk el az összes többi út fajtája és iránya esetében. Ez a függvény tehát az utak típusát, így a hozzárendelendő textúra meghatározását segíti. Végül meghívjuk az utak kirajzolásáért felelős függvényt.

4.4.5. Paraméterek határértékének átlépésének ellenőrzése

A lehelyezhető objektumok város paramétereit módosító számadatok vizsgálata során beállítottam a határértékeket, amelyek az egy perces időkorlát alatt még kiszámolhatók, és feloldhatók. Amennyiben bármelyik átlépi a megengedett tartományt, elindul egy számláló. Ez addig ketyeg, amíg a kiegyensúlyozatlan helyzet fennáll. Amennyiben a problémát sikeresen megoldottuk, a számláló nullázódik, a játék folytatható. Ha viszont az idő letelik, nem sikerült a problémát megoldanunk, egy képernyő válik láthatóvá, amely a játékteret eltakarva a játék végét jelenti.

4.6. Felmerült nehézségek

A fejlesztés során akadtak kisebb-nagyobb kihívásaim. Ilyen volt például az autók mozgatása során felmerült tört számok osztásánál keletkező, JavaScript-ben előforduló hiba. A tört számokkal végzett műveleteket ugyanis a nyelv nem kezeli le tökéletesen, ugyan azon művelet többszöri elvégzése egy számon nem ugyan azokat a hatásokat fejti ki. Ez a pici pontatlanság viszont a mozgatás sosrán végzett számításoknál jelentős probléma volt, így hosszas próbálgatás során sikerült kiküszöbölnöm a problémát.

4.7. Használt algoritmusok

4.7.1. Utak generálása

Ez a függvény paraméterben várja a lehelyezett objektum x és z koordinátáit. Minden esetben megvizsgáljuk, hogy a beértkező koordinátákat körülvevő párosokat már tartalmazza-e a pályán lévő objektumokat tároló asszociatív tömbünk. Amennyiben nem, hozzáadjuk őket a tömbhöz a megfelelő út irányát jelző számértékkel. Az utak irányait az x és z koordinátáinak műveleteivel kapjuk meg, azokhoz hozzáadunk vagy kivonunk belőlük egyet, így az eredeti koordinátát körbevevő 8 helyadatot megkapjuk.

```
if (!globals.map.has(`${x - 1}_${z - 1}`)) {
  globals.map.set(
    `${x - 1}_${z - 1}`,
    globals.groundObject.road.road_down_right
  );
}
```

4.2 kódrészlet: Jobbra lefele kanyarodó út számértékének út objektumhoz rendelése

Amint megvizsgáltuk az összes környező mezőt, meghívjuk az utak újraépítéséért felelős függvényt, valamint az utak összekötéséhez szükséges útpárosító segédfüggvényt a lehelyezett épület helyadataival.

4.7.2. Távoli utak összekötése euklideszi távolság számításával

Az utak összekötéséhez szükségem volt annak leellenőrzésére, hogy az adott épület lehelyezésével generált utaknak csak két szomszédja van, azaz előző utakkal nem került összekötésre, valamint hogy a pályán már van-e előzőleg lehelyezett út, távolabbi épületek már pályán léte miatt. Ezt az utak párosításának függvényében vizsgáltam.

Ez után hívtam meg a kezdő és végpontot választó segédfüggvényt, ahol meg kellett néznem, hogy az útjaink a többi úttal összekötve milyen távokat adnak ki. Ehhez a

$$d = \sqrt{((x_2 - x_1)^2 + (z_2 - z_1)^2)}$$

képletet használtam. Az így megkapott távolságokból a legkisebbet kiválasztva megkaptam a kezdő és végkoordinátákat az algoritmus folytatásához.

```
if (isRoad(value)) {
   let [xKey, zKey] = key.split("_");
   xKey = parseInt(xKey);
   zKey = parseInt(zKey);

   if (!((xKey == x + 1 || xKey == x - 1 || xKey == x) &&
      (zKey == z - 1 || zKey == z + 1 || zKey == z))){
    distance = Math.sqrt(
      (xKey - xCoordinate) * (xKey - xCoordinate) +
      (zKey - zCoordinate) * (zKey - zCoordinate));

    if (distance < min && distance !== 0) {
      min = distance;
      endX = xKey;
      endZ = zKey;
      startX = xCoordinate;
      startZ = zCoordinate;
    }}}
```

4.2 kódrészlet: legközelebbi végpontok kiválasztása

Ezután a kezdő koordinátától indulva folyamatosan lecsökkentettem a *z* koorináta értékét, amig a végkoordináta *z* tagjával egyenlőt nem kaptam, majd ugyan ezt elvégeztem az *x* taggal is. Így megkapva azon számpárok összességét, amik majd az új utakat fogják képezni. Ezeket a párokat hozzáadtam a map-hez, majd meghívtam az útgenerálás függvényt, amely megépítette az új utakat, valamint összekötötte őket a már meglévőkkel.

4.7.3. Járművek mozgatása

Az autók mozgatása egy animációként fogható fel. Ennek működéséhez szükség volt egy „requestAnimationFrame“ nevezetű beépített függvényre, amely közli a böngészővel, hogy animációt kívánunk végrehajtani, valamint megkéri a böngészőt, hogy hívja meg az általunk megadott függvényt, amely frissíti az animációt.

A függvényhívás után bejárjuk az autók tömbjét. Minden egyes egységen, azaz minden útszakaszon leellenőrizzük, hogy az autó pozíciója elérte-e az 50-et, azaz a fordulópontot. Ebben az esetben ha az autó nem tud már egyenesen menni, mert nincs előtte már út objektum, adunk neki egy új irányt. Abban az esetben, ha a járműtől jobbra és balra is út helyezkedik el, 70% eséllyel belra fordítjuk azt, különben jobbra folytatja tovább a mozgását. Ezzel az volt a célom, hogí csökkentsem az összeütközések számát, és kevésbé legyen kaotikus a forgalom, mégis legyen benne egy véletlen faktor. Amennyiben viszont csak egy irányba tud tovább haladni a jármű, úgy az egyetlen lehetséges koordinátát hozárendelve forgatjuk el és mozgatjuk azt tovább. Ezek után a pozíciót 1-re állítjuk, hogy a számlálás előröl induljon egészen 50-ig.

(ide tervezek még egy kódrészletet)

6. Telepítési és felhasználási útmutató

Tetszőleges fejlesztői környezetben megnyitva a programot, a terminálban az *npm run start* parancs kiadásával megjelenik a főoldal a böngészőnkben. A főoldalon angol nyelven olvashatunk egy rövid ismertetőt a játékról, valamint részletes leírást kapunk annak használatáról, képekkel demonstrálva azt. Rögtön a bemutatkozás alatt a "Let's play" gombra kattintva átnavigál minket egy betöltő oldalra, majd a játék megnyílik.

Bal oldalt találhatunk egy sávot, melynek tetején a város paraméterei találhatók. Ez alatt tudunk választani a lehelyezhető épületek és járművek közül. A képernyő további részén a játéktér található, melyen tetszőleges, a pálya méretei által korlátozott mezőkre navigálhatjuk a helyválasztó objektumunkat a nyílbillentyűk segítségével, valamint annak irányát is módosíthatjuk a space billentyű lenyomásával, így az épületünk lehelyezése előtt be tudjuk igazítani, hogy az merre nézzen. Egy épület lehelyezése után automatikusan generálódik köré egy úthálózat. Több épület lehelyezése esetén ezek az utak szintén a program által összekötésre kerülnek, az előző utak állapotai frissülnek annak függvényében, hogy alakultak-e ki körülötte kereszteződések, vagy sem. Amennyiben a pálya széléhez közelítünk, és épületeket helyezünk le, a játéktér mérete megnövekszik.

Lehetőségünk van járműveket lehelyezni, azonban ezeket csak utakra tudjuk megtenni. A jármű objektumok lehelyezés során menetirányba helyezkednek, így azok lerakásakor a forgatással és irányválasztással nem kell törődnünk.

A városunk megépítése során lehetőségünk van a már megépített részletek zavartalan megtekintésére, a teljes képernyős nézettel. Ebben a nézetben épületek lehelyezésére nincs lehetőség az oldalpanel eltüntetése miatt. Ez a mód csupán arra szolgál, hogy a zavaró elemeket eltünteti a látótérről. Ez nem egyenlő az F11 gomb hatásával elért teljes képernyő móddal, ugyanis a szerkesztési panel abban az esetben ugyan úgy látható és használható. Ezt a funkciót az egérrel való dupla kattintással tudjuk elérni, valamint kilépni belőle is így tudunk.

A statisztikai adatok helyes értéktartományban tartásának érdekében a lehelyezett objektumainkat okosan kell megválasztanunk. Amennyiben az értékeink túlzottan kilengnek valamelyik irányba, van egy percünk azt helyrehozni további épületek, járművek lehelyezésével. Amennyiben ez nem sikerül, a játékunknak vége. Az oldalon megjelenő "Try again" gombra kattintva a játékunk újra töltődik, és előről kezdhetjük az építkezést, vagy kiléphetünk a nyitó oldalra a “Go home” gombra kattintva.

7. Összefoglalás

(összefoglalás, végeredmény, mi valósult meg , motiváció, elékszített algoritmusok, milyen technológiákkal valósítottam meg)

A szakdolgozatom elkészítsét számos témakörre kiterjedő önfejlesztés előzte meg. A Three.Js ismereteim kibővítésére nagy mértékben szükség volt, így a dolgozatom közben több forrásból származó oktató videót néztem meg, hogy fejlesszem a tudásom ennek terén. Természetesen a modellezés iránti érdeklődésem nem volt elég a jó modellek készítéséhez, muszáj volt abba is időd fektetnem, hogy gyakoroljak, új trükköket, módszereket tanuljak a minél hitelesebb, esztétikusabb épületek elkészítéséhez. A dolgozatom témájának ellenére ez vette el talán a legtöbb időt, viszont ez is volt a legváltozatosabb, legpihentetőbb része a munkámnak. Maga a JavaScritpt tudásomon is volt mit csiszolni, voltak meglepő pontok a dolgozatom elkészítésekor, amikor nem tudtam, mi miért történik, nem voltam tisztában a nyelv alapvető bukkanóival, így ezeknek is részletesen utána kellett járnom, de legfőképp kísérletezés, saját kitapasztalás alapján tanultam a JavaScript esetén.

Munkálataim során kimaradtak egyes funkciók, amelyek bár nem képezték az elvárásaim részét, mindenképp hozzájárulnának a PolyCity élvezhetőbb, esztétikusabb játékká fejlesztéséhez. Ilyen például a forgalom irányítása, melynek során a járművek összeütközését megakadályoznám, a forgalom csak egy meghatározott irányba mehet, nem állhatnának fenn olyan helyzetek, hogy két autó egymással szembe halad, valamint kanyarodás során az elhaladó autók megvárnák egymást. Továbbá minél többféle épület, járműve lehetséges, annál változatosabb és érdekesebb lenne a játék, mind a statisztikai adatok alakulása, mind látványbeli szempontból, így a modellek állománya szinte a végtelenségig bővíthető lenne. Ezeken felül felmerült bennem a fejlesztés során, hogy a pénz, mint korlátozó tényező is bekötésre kerüljön, ezért esetleg a jövőben ennek implementálása is fejleszthetné a játékot, alakíthatná annak nehézségét.

(Ezt a részt is még kiegészítem)

Irodalomjegyzék

1. Legrövidebb út megkereséséhez használt segédvideó: https://www.google.com/search?q=count+way+between+two+points&oq=count+way+between+two+points&aqs=chrome..69i57.31489j0j1&sourceid=chrome&ie=UTF-8#kpvalbx=_FRBIY4-QBcSE9u8PgZ6syAQ_35
   utolsó megtekintés dátuma:
2. Hasonló játékok forrásainak gyűjtőoldala: https://www.makeuseof.com/tag/5-fun-online-city-building-games-run-browser/
   utolsó megtekintés dátuma:
3. Three.js journey
4. Udemy-Blender kurzus
5. Three.js jegyzet Számítógépes grafika gyakorlathoz :
   http://www.inf.u-szeged.hu/~tanacs/threejs/index.html
   utolsó megtekintés dátuma: 2023.01.15.
6. Threejs hivatalos honlapja: https://threejs.org/
   utolsó megtekintés dátuma: 2023.01.10.
7. HTML elemek elrejtése:
   https://www.w3schools.com/howto/howto_js_toggle_hide_show.asp
   utolsó megtekintés dátuma:
8. Logók és betöltőképernyő tervezéséhez: https://www.autodraw.com/
   utolsó megtekintés dátuma:
9. Visszaszámláló a játék végéhez:
   https://www.w3schools.com/howto/howto_js_countdown.asp
   utolsó megtekintés dátuma:
10. requestAnimationFrame pontos leírása: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame
    utolsó megtekintés dátuma:

(A megtekintések dátumát, valamint a hivatkozásokat még nem tettem bele)

Nyilatkozat

Alulírott ……..…… szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet …….. Tanszékén készítettem, …….……. 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 / diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

Dátum

Aláírás

Köszönetnyilvánítás