Budapesti Műszaki és Gazdaságtudományi Egyetem

Építészmérnöki kar

Morfológia és Geometriai Modellezés Tanszék

Tudományos Diákköri Dolgozat

Beépítési koncepciók evolúciója

Készítette:

Vincze Krisztián

Konzulens:

Dr. Fehér Eszter

2023

Tartalomjegyzék

Bevezetés

Beépítési koncepciók tervezése

Az építészeti tervezés bonyolult és időigényes feladat. A tervezett épületnek több előre meghatározott problémakörre kell optimális választ adnia. A tervezés egy próbálgatással folytatott iteratív folyamat, mely során az építész a különböző szaktervezők bevonásával fejleszti a tervet, hogy az egyéni igényeinek megfelelő, hatékony eredményt érjen el. Ez a holisztikus szemlélet szükséges ahhoz, hogy jól működő házakat, városokat építsünk.

A várostervezés egy alfeladata a beépítési koncepciók tervezése (1. ábra). Egy város léptékű beépítés elrendezésének számos lehetősége lehet. A befolyásoló tényezőket, a környezetben lévő elemeket és létrejövő épületeket önálló, de még is egymástól függő elemekként lehet tekinteni. Az utóbbiak létrejötte, mérete és pozíciója is függ a többitől. A próbálgatási folyamat lényege, hogy méret és pozícióváltoztatgatás útján hozzunk létre a szempontjainkra optimálisan kialakított beépítéseket, aminek ellenőrzése és kiértékelése a változtatgatási folyamat része. A tervezés során hagyományosan csak néhány elrendezési variációt hoznak létre, amelyek megfelelnek a kiindulási feltételeknek, és ezeket fejlesztik tovább.

(1.ábra) Város léptékű beépítési koncepció egy budapesti területre

Meglévő módszerek

A közelmúltban megjelent több olyan publikáció, amely célja, hogy az urbanisztikai tervezésben megjelenő manuális iterációt automatizálja a generatív tervezés eszközeivel. Ezek egyrészt lehetőséget adnak arra, hogy a tervvel szemben támasztott követelményeknek megfelelő geometriákból sok száz variációt hozzunk létre, másrészt arra is választ adnak, ha a kiinduló feltételek ellentmondásosak és nem létezik megoldás. Kézi eszközökkel ezeknek az alapvetően geometriai jellegű optimalizációs feladatoknak a megoldása időigényes. Az elrendezés és így a koncepcióalkotás is hatással van arra, hogy a beruházás elérje az elé támasztott célokat. A generatív tervezés lehetőséget ad, hogy felgyorsítsuk a tervezési folyamatot. Minden beruházásnál más szempontrendszert állítanak fel, minden környezet és probléma egyedi. A sok különböző szempont egymásra hatásának mechanizmusa előre nehezen látható. Ha meg tudjuk fogalmazni a célokat és minden cél meghatározásához használt eljárás ismert, akkor lehetséges a beépítési koncepciók optimalizálása. A generatív koncepcióalkotás lényeges előnye a több szempont szerinti optimalizálás [1][2], ezzel elérhető lenne a holisztikus, teljesítmény alapú tervezés már a legkorábbi fázisokban. Ezt követi több létező projekt is. A Delve Sidewalk labs ingatlanfejlesztési cégeknek szolgáltat egy tervezői platformot, amivel költségbecslést lehet készíteni amellett, hogy több szempont szerint szimulálja a beépítést, mint a bejárhatóság, benapozás, kapcsolódás a közlekedési hálózathoz vagy elérhető bérelhető alapterület felbecslése [3]. Sokféle megközelítés alakult ki a formák generálására az utóbbi évtizedekben. A megközelítések változásával egyre több lehetőség nyílt építészeti vagy városépítészeti problémák megoldására is. A változást az hozta, hogy a kezdeti cellákon alapuló formaképzést [4] felváltotta az önálló és módosítható geometriák képzése. Több önálló formanyelv jött létre [5]. Valamely esetben ennek leképzésére sajátos programozási nyelvet is készítettek [6]. Mostanra építészeti vagy városépítészeti megközelítésen alapuló formanyelvek is kialakultak, amikben a generálás valamilyen funkcionális sémán [7], vagy sűrűségi és tipológiai összefüggéseken alapulnak [8]. Ezek a rendszerek összességében úgy alkothatnak jó eredményt, ha figyelembe veszik a különböző építészeti elemek kapcsolatait, a használatból és funkciókból eredő térigényeket. Képesnek kell lenniük a geometriák módosítására úgy, hogy közben kövessék ezeket a feltételeket, és egyszerre számos egyéb befolyásoló tényezőre is optimális eredményt adjanak, mint például a környezeti elemzések és a beruházói igények.

Evolúciós eljárás

Christopher Alexander építészetről és tervezésről szóló munkássága világított rá arra, hogy az építészet felfogható egymással interakcióban lévő, egyedi tulajdonságú objektumok összességeként. Munkája nagy hatással volt az ún. objektumorientált programozás kialakulására. A mintázatokat érintő kutatása [10][11] ihlette a fejlesztőket abban, hogy a szoftvereket objektumokra alapozzák függvények helyett [9].

Míg az irodalomban megtalálható módszerek többségének [7][8][14][15] alapelve, hogy előre definiált blokkokat helyeznek el a telken, melyeknek a pozícióját változtatják és optimalizálják, jelen dolgozatban bemutatott algoritmus a hagyományos tervezéssel analógon módon a beépítést alkotó objektumok evolúcióján alapul. A tervezés az épületek körülbelüli helyének a meghatározásával indul, a végső geometria pedig fokozatosan alakul ki. A beépítés eltérő tulajdonságokkal rendelkező és egymással kapcsolatban álló objektumokból épül fel, melyek vizsgálják környezetüket és az átfogóbb egységet, majd ennek megfelelően változtatják önmagukat.

Módszertan

Alapfogalmak

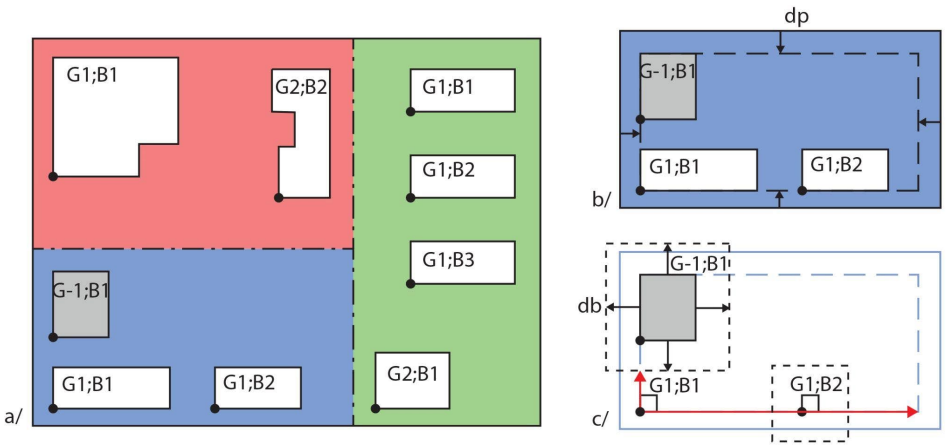
A beépítési koncepción egy telket és az azon elhelyezett épületeket értjük. A telek egy sokszög, amelyet utak darabolhatnak telekrészekre. A telekrészekhez mindig tartozik egy földhasználat, amely definiálja, hogy az adott telekrészen az épületeknek milyen követelményeknek kell megfelelniük. Az épületek csoportokba rendeződnek, a csoportok egyedi tulajdonságokkal rendelkezhetnek. Az épületcsoportok hozzá vannak rendelve valamelyik telekhez ill. telekrészhez, így a csoportok és az őket alkotó épületek mindig az adott telken ill. telekrészen belül helyezkednek el. Az épületek is sokszögek, melyeknek egy kitüntetett csúcsa az adott épület referenciapontja. Az épületek egymástól legalább db, a telek szélétől legalább *dp* távolságra helyezkedhetnek el. Ezek alapján meghatározható a teleknek és az épületnek is egy *dp* ill. *db* vastagságú tolerancia zónája, amelyen belül nem helyezkedhet el másik épület. A tolerancia zóna a telket ill. épületet alkotó sokszög befelé ill. kifelé történő párhuzamos eltolásával határozható meg. A felhasználó által megadhatók akadályok, amelyek olyan sokszögek, amelyek mérete és pozíciója az evolúció alatt nem változik és az épületek nem metszhetnek bele ezek toleranciazónájába. Ezen akadályok lehetnek pl. meglévő épületek, tereptárgyak vagy tervezésen kívül eső területek. Az evolúciós algoritmus az épületek oldalait

ezen korlátozások figyelembevételével növesztik. Egy adott oldal növesztése mindig az oldal normálisának irányában, kifelé történik. Az alapfogalmak az 2. ábrán láthatók.

(2. ábra) *a)* Az ábrán látható egy belső útvonallal meghatározott telekfelosztás. Az ábra három telekrészt mutat, amihez különböző színnel jelölt földhasználatok tartoznak. A fehér színű sokszögek épületek, míg a szürke sokszög egy akadály. A telekrészeken *Gi*-vel jelölt épületcsoportok vannak, az azonos sorszámú épületek tartoznak egy csoportba. A G*1*-es jelzés az akadályokat tartalmazó csoportot jelöli. A *Bi* jelzés az épületek sorszámait jelöli. Az épületek megjelölt csúcsa az adott épülethez tartozó referencia pont. *b)* Az ábrán az egyik telekrész látható a telek *dp* távolsága által meghatározott toleranciazónájával. *c)* Az ábrán két épület és egy akadály látható. A fekete szaggatott vonalak az épületekhez tartozó *db* szélességű tolerancia zónákat jelölik. A világoskék vonal a telek, a világos kék szaggatott vonal pedig a telekhez tartozó tolerancia zóna határa. A *G1;B1* épület esetében piros nyilak jelölik a lehetséges növekedési irányokat és távolságokat. Az akadály felé csak a tolerancia zónáig nőhet, a másik irányba viszont a telekhatárig, mivel a két épület azonos csoportban van.

Algoritmus

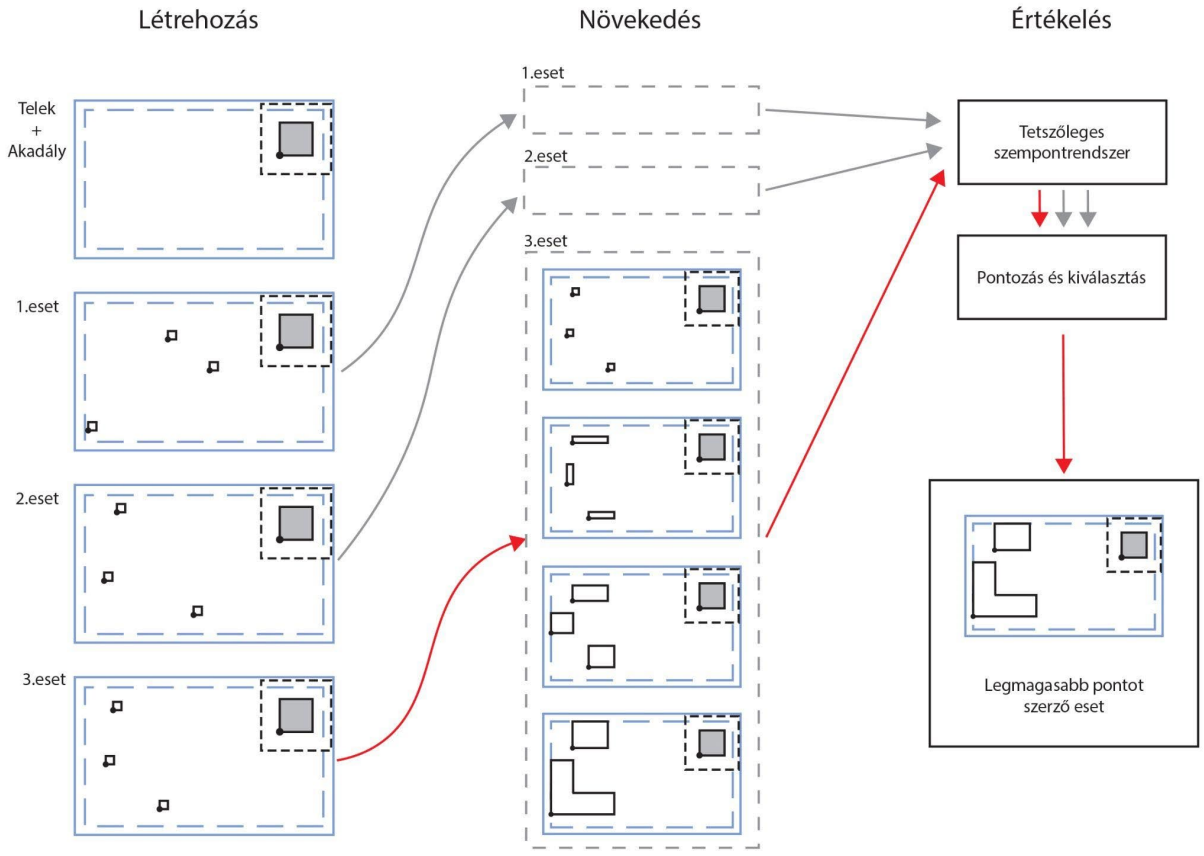
Az algoritmus fő célja, hogy beépítési koncepciókat generál és keresi a felhasználó által megadott korlátoknak megfelelő és valamilyen értékelési szempont szerint kedvező megoldásokat. A beépítés, vagyis a telken lévő épületek és útvonalak elhelyezése az épületek növesztésével és pozicionálásával megy végbe, mely folyamat a felhasználók által megadott paramétereken alapul.



A program bemenetei a telek geometriája és a beépítést alkotó további elemek (telekrészek, telekhasználat, csoportok, épületek) tulajdonságai és velük szemben támasztott követelmények. A beépítést alkotó elemek önálló objektumok és hierarchikus struktúrában vannak elhelyezve. Minden hierarchikus szint önálló objektumként van kezelve. Minden szint felelős az alatta lévőért. Az épületek növekedhetnek és változtathatják pozíciójukat, de parancsot, a változtatási

lehetőségekre korlátot a magasabb hierarchikus csoportok objektumai, és a saját belső paramétereik adnak.

Az algoritmus első lépése, hogy kis méretű, négyzet alakú épületeket helyez el random módon a telken belül úgy, hogy azok már kezdetben is megfeleljenek a velük szemben támasztott követelményeknek. Az evolúció során minden időlépésben növekszik minden épület egy-egy random kiválasztott oldala a hozzá tartozó normálvektor irányában. Az objektumok növesztés előtt információkat gyűjtenek a környezetükről, amivel megismerik a rájuk vonatkozó korlátokat. A növesztés nagysága egy nulla és a lehetséges maximális növesztések közötti random szám. A folyamat addig tart, míg a beépítés valamely mutatója el nem ér valamilyen célfüggvényt (pl. beépítési százalék). Az algoritmus lehetőséget ad arra, hogy a növesztést minden iterációban kiértékeljük, az aktuális geometrián szimulációkat végezzünk és ezeknek megfelelően, a forma és az elhelyezkedés változtatásával optimalizáljuk a geometriát. A generálási, növesztési és értékelési folyamatot a 3. ábra mutatja be.



(3.ábra) Az ábrán látható folyamatábra az algoritmus lefutásának lépéseit mutatja be. A generátor meghívásával, a bemeneti paraméterek alapján létrejön a telek (folytonos világoskék vonal) és tolerancia zónája (világoskék szaggatott vonal), az akadályok (szürke téglalap) a tolerancia zónájukkal (fekete szaggatott vonal). Ezután létrejönnek a beépítési esetek (elrendezési variációk), az épületek referenciapontjai és a kezdeti geometria. Az

összes objektum létrejötte után megkezdődik az iteratív növekedés a leállási feltétel teljesüléséig. A létrejött eseteket értékeljük tetszőleges kiértékelő függvényekkel, majd a legjobban teljesítő eset adjuk vissza és rajzoljuk ki.

A program részei

A következőkben részletesen bemutatom az algoritmust megvalósító Python kód részeit, hierarchiáját, a felhasznált objektumokat és azok működését. A magyarázatot a *Generator()* objektum meghívásnál végbemenő folyamat leírásával zárom. A növesztés részletei és a szükséges elméletet az ezt követő fejezetben foglalom össze.

A beépítést alkotó objektumok

A legalapvetőbb objektum a *Vector_fp()*, mely egy *Numpy array*-en alapuló vektort hoz létre bemeneti pontokból és a funkciói segítségével aritmetikus műveleteket végezhetünk egyéb vektorokkal. A vektorok az épületek alapelemei, de egyéb műveletek elvégzésekor is hasznukat vesszük. Az épületek módosításának logikája vektorműveleteken és trigonometrikus műveleteken alapul. Ezek segítségével egyszerűen elvégezhető az épületek növesztése és a környezetük feltérképezése.

A másik fontos alapelem a *Polygon()*, vagy poligon osztály. Ez egy ősosztály, amelyből származtatunk más osztályokat. Azért definiáltuk külön, mert több különböző objektum módosításához is szükség van ugyan azokra a funkciókra. Itt szerepelnek a poligonális objektumok pontjainak, oldalainak, oldalak darabszámának, területének, középpontjának meghatározó és növesztéséért felelős függvényei. Ezek a függvények olyan attribútumokat használnak, amik származtatott osztályokban találhatók meg.

A *Site()*, vagyis telek osztályt a *Polygon* osztályból származtatjuk és vektorokból épül fel. Létrehozásakor a felhasználó által megadott feltételek segítségével automatikusan kiszámítja a tolerancia zónáját, ami a vektorainak arányos csökkentéséből áll elő, ezt a csökkentett méretű sokszöget a benne létrejövő épületek nem metszhetik. A *Polygon* függvényei hívhatók, emellett az épületekhez tartozó pontok meghatározásakor lehet lekérni a telek szélességi határértékeit minden oldallal párhuzamos irányban.

A *Building()* osztályt ugyancsak a *Polygon* osztályból származtatjuk és az épületeket reprezentálja. Meghívhatók a *Polygon* osztály függvényei és emellett beállítható a tolerancia zónája, amit frissíteni kell minden növesztés után, hogy megfelelően működjön a metszések

ellenőrzése más épületek növesztése előtt. Az épületet lehet még forgatni, vagy megtagadni a növesztési lehetőségét. Az utóbbival állíthatók be a fix épületek vagy akadályok, de akár a generált épületek növesztése is korlátozható.

A *Group()*, az épületcsoport osztályt foglalja magába és hozza létre a hozzá tartozó épületeket a referencia pontjaikkal együtt. Egy telekrészhez több csoport is tartozhat. Ennek több oka is van: egyrészt így nő egy telekrész beépítésének variációja, másrészt egy azonos csoport épületei pozícionálisan összefügghetnek, főleg ha a pontok elhelyezése nem random. Minden csoport épületeinek pontjai azonos telek oldalhoz igazítva lesznek kiosztva. Ebből adódóan egy szigorú és egységes pontkiosztás után, egy csoport pontjai szorosan sorakoznak majd az adott telekoldal mellett, ezt a 4. ábra szemlélteti. Másik fontos aspektusa egy csoportnak, hogy a hozzá tartozó épületek összeolvadását is megengedheti, miközben a különböző csoportok épületeivel nem nőhetnek össze.

A *Landuse()*, vagyis földhasználat osztály, mely a telken jelen lévő különböző földhasználatokat hivatott megkülönböztetni. Ha nincs több részre osztva a telek, akkor csak egy ilyen objektum létezik. Ez felel a telekrészen létrejövő csoportokért, épületekért és akadályokért. További kutatás során be lehet vezetni a földhasználati kategóriákat és az ennek megfelelő generálást, ami tér szintaxis [12] alkalmazásával teljesíthető.

Vezérlés

A *Case()*, azaz eset osztály foglal magába egy generálási esetet a megadott paraméterekre és telekre. További fejlesztés során itt lehet bevezetni egy belső közlekedő rendszer kialakítását, ami telekrészekre osztja a telket, ezek alapján hoz létre földhasználati objektumokat, amik pedig minden más alsóbbrendű objektumot.

A *Generator()* osztály felelős az összes objektum létrehozásáért és a rajtuk végzett módosítások kezeléséért. A növesztés parancsokat itt hívjuk meg és minden iterációban ellenőrizzük a további növesztés szükségességét. Akkor fejezzük be a növesztést, ha az iterációk száma elér egy határt, vagy a beépítés elér valamilyen célfüggvényt pl. a beépített terület eléri az előre meghatározott értéket. Ha teljesül a feltétel, akkor a következő lépés a beépítési esetek értékelése lesz.

A kezdeti geometria létrehozása

A *Generator()* osztálynak három bemenete van. Egy paraméter lista, a telek és a hozzá tartozó akadályok. Az előbbinek a formátuma *dictionary*, vagyis szótár. A következő paraméterek tartoznak hozzá:

params = {

"max_area_ratio": 0.3,	Maximális beépítési arány
"maxit": 100,	Iterációk maximális száma
"max_growth": 5,	Oldalak maximálisan megengedett növekedése [m]
"num_cases": 10,	Generált esetek száma
"ngroups": 2,	Csoportok száma
"ngbuildings": 3,	Épületek száma csoportonként
"init_building_size": 1,	Alap épület oldalhossza [m]
"dist_from_site": 5,	Távolság tartandó a telektől [m]
"dist_from_building": 1,	Távolság tartandó az épületektől [m]
"group_intersection_allowed": False,	Csoporton belüli összenövés engedélyezése
"keep_history": False,	Generálási előzmények megtartása
"grow_type": 1}	Növesztés típusa [0,1]

A *Generator.base()* funkció meghívásával létrejön egy kezdeti elrendezés és minden szükséges objektum a következőkben bemutatott sorrendben. Az esetek létrehozzák a *Site()* - telek és a *Landuse()* - földhasználat objektumokat. Az utóbbiak felelősek a funkciók szétosztásáért, épületcsoportokért és a morfológiáért. Az esetekben kerül meghívásra a Landuse összes funkciója, mellyel létrejönnek az épületcsoportok, és azokon belül a referencia pontok és az ezeknek megfelelő épületek és az akadályok.

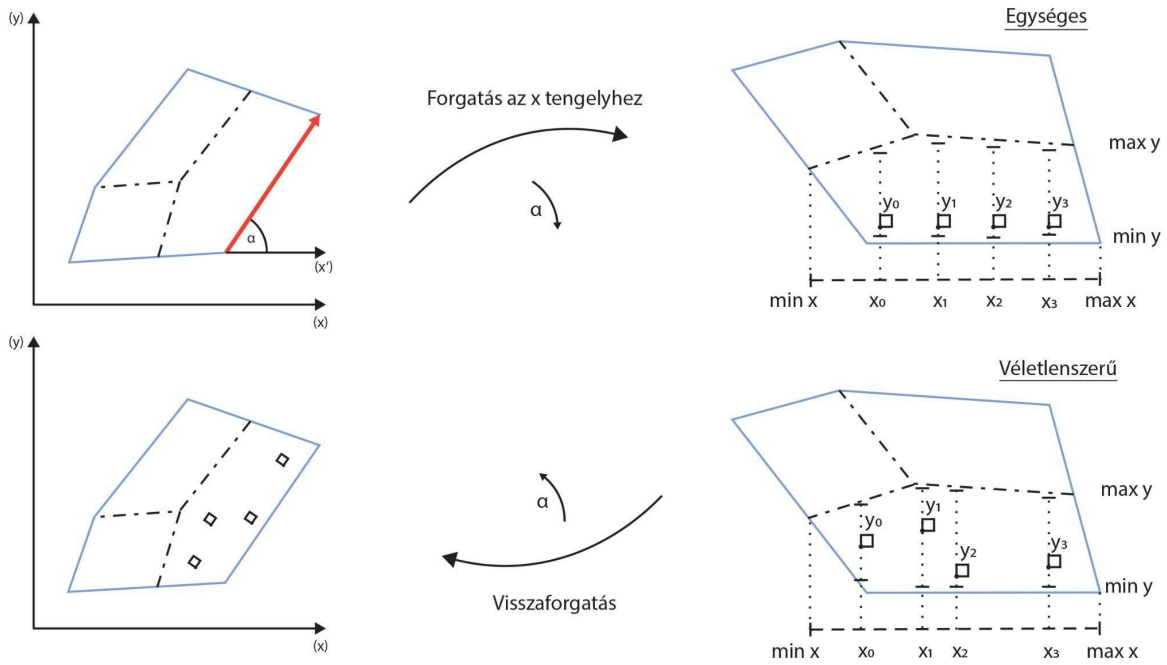
A referenciapontok jelentik az épületek kezdőpontjait. Meghatározásuk csoportonként történik. Minden épületet pozícionálhatunk úgy, hogy a telekhatárának valamely oldalával párhuzamos legyen. Egy csoporton belül pedig minden épület referenciapontját azonos telekoldalhoz

igazíthatjuk, emiatt a csoport önálló elrendezési szabállyal rendelkezhet. Ilyen például egy sorházas beépítés, ahol az épületek párhuzamosan, egyenlő távolságban helyezkednek el, és mind ugyanahhoz a telket határoló útszakaszhoz illeszkednek.

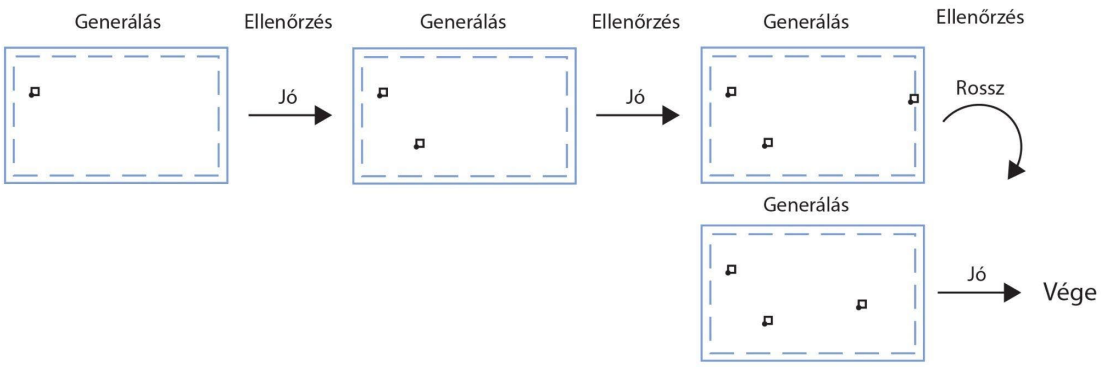
Ennek megfelelően a pontok létrehozásnál szükség van egy kiválasztott telekoldalra és a kívánt darabszámra. Jelenleg a csoportok és az épületek száma kívülről megadandó adat, ezt további fejlesztés során módosítani lehetne úgy, hogy minden telekhasználatnál az algoritmus eldönti a szükséges csoportok és azon belül az épületek mennyiségét figyelembe véve a megadott földhasználatot, környezetet, funkciót és tipológiát. Vagy egy másik megközelítés szerint csak a telekadottságok alapján, funkcionális és használati alapvetések és szükségek figyelembevételével hoz létre beépítést nem meghatározott számú csoporttal és épülettel, majd utólagos klasszifikáció során meghatározza a lehetséges földhasználati kategóriát és funkciót. Ezen funkciók fejlesztése további kutatást igényel.

A pontok megválasztása úgy történik, hogy addig forgatjuk a telket, amíg a kiválasztott telekoldal párhuzamos lesz a koordináta rendszer x tengelyével. Ez alapján meghatározzuk a kiválasztott oldalnak megfelelően a telekhatár minimális és maximális x koordinátáit. Felosztjuk ezt a távolságot méterenként és minden osztásköz egy potenciális x koordinátája lehet a létrehozandó pontoknak. Ezekből kiválasztunk egyet és a rá állított y tengellyel párhuzamos egyenes és a telek metszéspontjai fogják meghatározni az x koordinátához tartozó minimális és maximális y koordinátát, ami között ki is választunk egy értéket. Ezzel létrejött egy pont, melyet akkor vehetünk fel az épületek referenciapontjainak listájára, ha az nem esik bele egyik fixen elhelyezett akadályba, vagy a pontra illesztendő alap épület nem ütközik a telekhatárral, más épülettel vagy akadállyal. Az alap épület méreteit szintén kezdeti paraméterként adhatjuk meg. Ez a folyamat addig ismétlődik, amíg az előre megadott pont mennyiség létrejön. Miután minden pont létrejött, visszaforgatjuk őket a telekkel az eredeti pozícióba. Ezeket a folyamatokat a 4-es és 5-ös ábra szemlélteti.

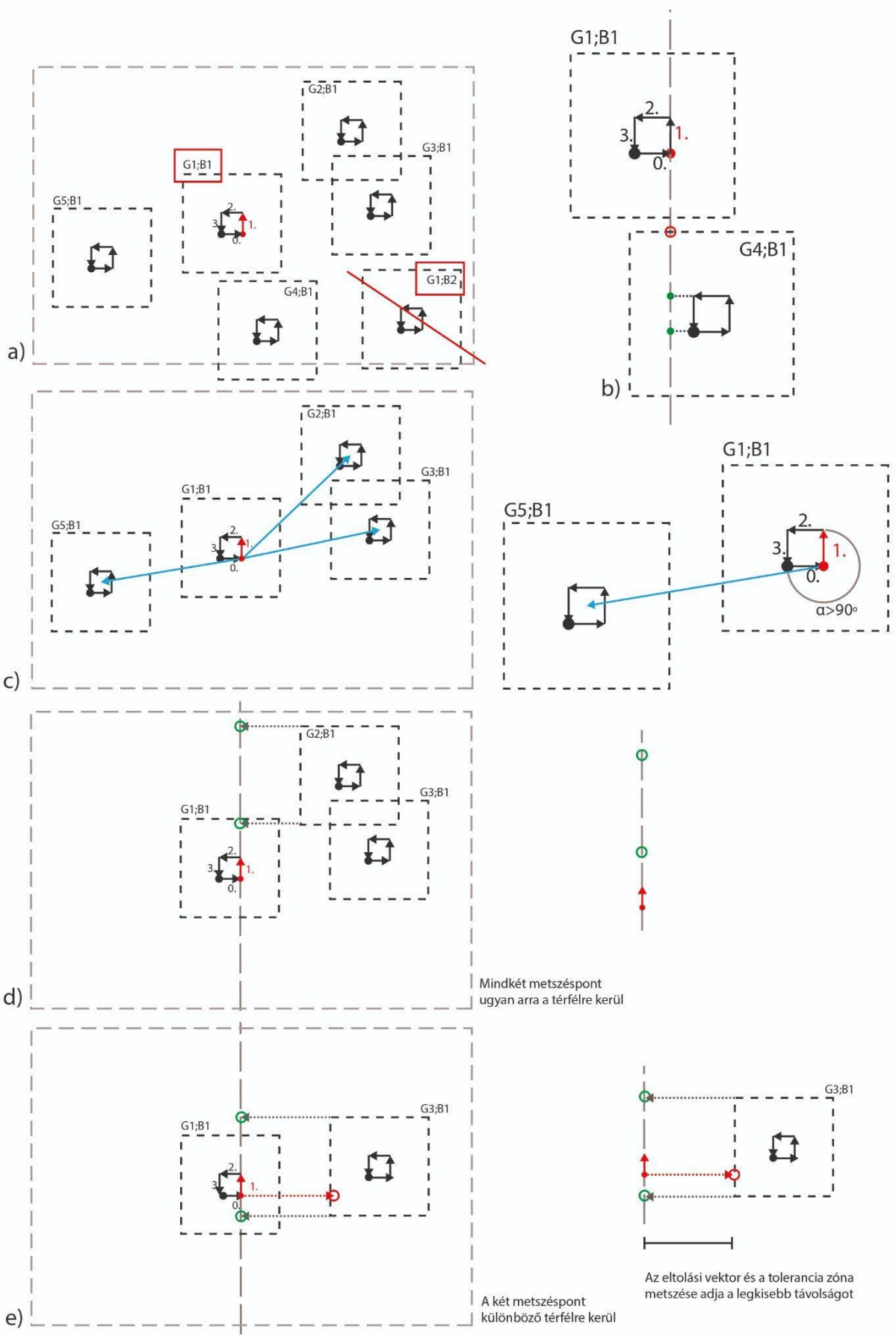
Ezután létrehozhatók az épületek, mellyel létrejön a Generator objektum. A következő lépésekben módosításokat hajthatunk végre a létrejött épületeken.



(4.ábra) Az ábra a pontgenerálási folyamatot mutatja. A tetszőleges formájú telek egyik oldalát kiválasztjuk és az általa meghatározott vektor és az x tengely által bezárt α szög szerint elforgatjuk a telket. Az adott telekrészen lemérjük a x koordináták határait, ezen belül generálunk az adott csoport épületszámának és a kívánt pontelrendezésnek megfelelő koordinátákat, majd meghatározzuk egyenként az y koordináták határait és közte az előbb említett feltételek szerint felveszünk minden helyre egy értéket. Ha létrejövő pontokat α szöggel visszafelé forgatjuk, megkapjuk a végleges pontokat.



(5.ábra) Az ábra a pontok kiválasztásának folyamatát szemlélteti. Akkor vehetünk csak fel egy pontot, ha annak a tolerancia zónája nem metsz bele a telek, már felvett pont vagy az akadályok tolerancia zónájába. Ha ez teljesül, felvesszük a pontot, ha nem, akkor újra generálunk. Addig ismételjük a folyamatot, amíg előáll a kívánt pontmennyiség.


a)
G1;B1
G2;B1
G3;B1
G5;B1
G4;B1
G1;B2
2.
3.
1.
0.
b)
G1;B1
G4;B1
c)
G1;B1
G2;B1
G3;B1
G5;B1
α>90°
d)
Mindkét metszéspont
ugyan arra a térfélre kerül
e)
A két metszéspont
különböző térfélre kerül
Az eltolási vektor és a tolerancia zóna
metszése adja a legkisebb távolságot

(6.ábra) Az ábra az elmozgatásra váró oldal környezetének vizsgálatát szemlélteti. A G1;B1 épület 1-es sorszámú oldalát szeretnénk mozgatni. Az épületeket a vektoraikkal ábrázoltuk, az épületek körüli szaggatott vonal a hozzájuk tartozó tolerancia zónát jelöli. a) Először kizárjuk a G1;B1-el azonos csoportban (G1) lévő épületeket, ha az összenövés megengedett. Ezután sorban haladunk a további épületeken. b) Példa egy vizsgálatra: a G1;B1 és a G4;B1 épületek viszonyának vizsgálata. Először egyenest állítunk a vizsgált oldalra. Ha metszést találunk egy tolerancia zónával (piros kör), akkor megvizsgáljuk, hogy az objektum oldalainak vetített képe hova kerül az egyenesen (zöld körök), ha mindkét pont egy térfélre esik az eltolandó oldalhoz képest, akkor kizárjuk a vizsgálatból. c) Ezután a vizsgált oldal normálisának és a többi épülettel összekötött vektornak az egymással bezárt szögét vizsgáljuk. Azok az épületek nem eshetnek a növesztés irányába, amelyeknél ez a szög nagyobb, mint 90 fok, ezért ezeket kizárjuk a vizsgálatból. A maradék épületeket vagy akadályokat egyenként vizsgáljuk. d) A G1;B1 és a G2;B1 épületek viszonyainak vizsgálata. A G2;B1 oldalait rávetítjük a kiválasztott oldalra illeszkedő egyenesre. Ha a vetített pontok egy térfélre kerülnek, akkor kizárjuk az adott épületet a vizsgálatból. e) A G1;B1 és a G3;B1 épületek vizsgálata. Ha a vetített pontok közül legalább egy a vizsgált oldalra esik, vagy különböző térfélre esnek, akkor az eltolási vektor és a tolerancia zóna metszése által meghatározott távolsága és a tolerancia zóna pontjainak a vizsgált oldaltól mért távolsága közül a kisebb lesz a megengedett maximális növesztési távolság.

Növesztési lehetőségek megállapítása

Azt az alapfeltételt határoztam meg az épületek módosítására, hogy az azonos funkcionális kategóriában lévő épületek összenőhetnek és egyesülhetnek, viszont a különbözőknek meghatározott távolságot kell tartaniuk egymástól. Egyik épületcsoportban lévő épületek azonos funkciót kapnak, így kézenfekvően az egy csoportba tartozás ellenőrzésével meg lehet állapítani ezt a kötöttséget. Viszont az összenövés lehetőségét előre meg kell adni a paraméterek között.

Mielőtt egy épület oldalait növeszthetnénk, információkat kell gyűjteni az épület környezetéről. Meg kell állapítani, hogy a megnyújtandó vektor vagy az eltolásra kerülő oldal milyen távolságban ütközik egy másik objektum tolerancia zónájával, ami egy másik objektumhoz, épülethez, fához, vagy a telekhez tartozik. Épületekre vonatkozóan többféle, pontosabban 10, 15 és 20 méteres távolságtartással is lefuttattuk a programot. Fáknál ezt 5 méterre állítottuk, fix épületeknél pedig 10 méterre. A maximális növesztés viszont nem feltétlen a tolerancia zóna határáig tartó távolság, hanem egy előre megadott paraméter a *max_growth*. Ez az érték lesz a növesztés maximális távolsága, ha a mért távolságok nagyobbak, ellenkező esetben a legkisebb mért távolság lesz a növesztési maximum.

Növesztési távolságok megállapítása

A növesztés megkezdésekor ki kell választanunk azt az oldalt, amit el szeretnénk tolni, ez után az határozzuk meg a növesztés mértékét, ami lényegében az eltolás mértéke. Azért beszélünk még is növesztésről, mert a kiválasztott oldalhoz tartozó vektor szomszédos két vektorát az eltolás mértékével kell meghosszabbítani a fentebb leírtak szerint.

Több szempontból meg kell vizsgálnunk az oldal környezetét, ha meg akarjuk kapni a minimális távolságot. Első körben ki kell zárnunk a vizsgálatból azokat az épületeket, amik az oldalra állított egyeneshez képest, a növesztés irányával ellentétes oldalon vannak, ugyanis ezek biztosan nem ütközhetnek az eltolt oldallal. Az eljárásunk szerint először megvizsgáljuk, hogy az eltolandó oldalra állított egyenes lehetséges metszéseit a többi épülettel, ha van ilyen épület, akkor ezeken további ellenőrzéseket is kell végezni. A maradék épületek közül a kizáró feltételt egy egyszerű ellenőrzés adja. Meg kell határoznunk két vektor által bezárt szöget, amit a növesztés irányú egységvektor és az eltolt oldal egyik pontja és a vizsgált épület geometriai középpontja alkot. Ha ez a szög nagyobb, mint 90 fok, akkor a vizsgált épület biztos a növesztéssel ellentétes oldalon van, ezért nem kell tovább vizsgálni. A további vizsgálatok során minden épület oldalain iterálva azt vizsgáljuk meg, hogy az adott oldal pontjainak az eltolt oldalra vetített képe milyen pozícióba kerül. Ezt az eltolandó oldal vektorának, és az oldal kezdőpontja és a vizsgált oldalak pontjaihoz húzott vektorok skaláris szorzatával ellenőrizzük. A számítások alapján meghatározhatjuk, hogy a vizsgált épület oldalának vetített képe az eltolandó oldal pontjai közé esik, vagy bentre kerül. Másik lehetőségként a vetített oldal egyike bentre, a másik kintre esik. Ezeknél a lehetőségeknél kell a pont és oldal távolságát meghatározni, majd a legkisebb távolság lesz a maximális növesztési lehetőség. Ha nem találunk metszést, akkor az eltolandó oldal két pontjára állított vektor metszését kell keresni a telekkel. Így mindenképp visszakapunk egy maximális növesztési távolságot. A távolságok vizsgálatát a 6. ábra szemlélteti.

A növesztés algoritmusa

Az algoritmus jelenlegi állapotában kétféle módosítási lehetőséget tettünk lehetővé. Lehetséges egy kiválasztott oldal vagy az összes oldal egységes növesztése is a lehetséges irányokban és a maximális növesztési távolságon belül. Az oldalak hosszát módosíthatjuk egységesen, vagy egy oldal eltolásának megfelelően. További fejlesztés során számos egyéb módosítási lehetőséget is érdemes beépíteni. Ilyen a referenciapont elmozgatása, elforgatás, több oldalt közrefogó növesztés, újabb oldal beillesztése az épületbe és az épület tulajdonságok lemásolása.

Az épület a legfontosabb egysége a generálásnak, ez a legnagyobb rendű önálló geometriával rendelkező objektum. A növesztés során hajt végre módosításokat a generátor az épületeken úgy, hogy valamelyik beépített függvényét hívja meg. Ezek a függvények módosítják az épületek pozícióját, elfordulását és oldalhosszait. Minden módosítás egyszerű geometriai műveletek útján elvégezhető, ami az épületek felépítésének köszönhető.

Egy épület két alapelemből állítható elő, egy referencia pontból, és egy abból induló vektor láncból. A vektorlánc egy nem felcserélhető sorrend szerinti lista, aminek elemeit sorban a referencia ponthoz adva az épület többi pontját kapjuk meg. Ha az összes vektort összeadjuk nullvektort kapunk. Ez a felépítés könnyen számolható növesztést tesz lehetővé. Egységes növesztés érhető el, ha az összes vektor hosszát ugyanolyan mértékben növeljük és a referencia pontot eltoljuk, úgy, hogy az alakzat középpontja helyben maradjon.

Adott oldal megnyújtása érhető el, ha az oldalnak megfelelő vektor, és a listában következő két szomszédos vektor által alkotott láncot vizsgáljuk. Ebben az esetben az eredeti állapot első vektorának kezdőpontja és az utolsó vektor végpontja fixen maradó pontok és az első vektor tetszőleges növekményéhez kell kiszámolni a másik két vektor megfelelő hosszát, ami szinusz tétellel lehetséges a vektorok szögtartását feltételezve. Ez a fajta növesztés bonyolultabb, tetszőleges szögekkel rendelkező poligonok esetében is alkalmazható. Több közrefogott oldal eltolására is alkalmazható.

Bármilyen módosítás után frissítjük a vektorláncot, amiből a pontok, oldalak, vagy a tolerancia zóna számítható. Emiatt akár új vektorokat is elhelyezhetünk a láncban a megfelelő számítások elvégzésével, vagy teljes mértékben kicserélhetjük a vektorláncot. Így az objektum megtartásával végezhetünk módosításokat egyszerűen és rugalmasan. A vektorlánc iterációnként keletkező adatainak megtartásával pedig bármely múltbeli állapot visszaállítható.

Épületek összevonása

Amikor két épület között megengedett az összenövés, akkor a távolság számításakor nem veszik egymást figyelembe, ezért ezek egymásra nőhetnek. Minden iterációban ellenőrizzük az épületek egymásra takarását és ha a takarás elér egy bizonyos mértéket, akkor a két épületből egyet hozunk létre. Ez úgy történik, hogy az egymásra takart területet összehasonlítjuk mindkét épület alapterületével és ha ez egy előre megadott százalékot elér, akkor végrehajtjuk az egyesítést. Az egyesítés a *Shapely* modul segítségével érhető el. A megfelelő geometriák létrehozása után ezek metszete meghatározható. Ha a feltétel teljesül, akkor létrehozhatjuk a

két geometria unióját, ami visszaadja az egyesített geometria pontjait. Ezekből a pontokból létrehozhatunk egy vektorláncot, amire pedig kicseréljük az egyik épület vektorláncát és ennek megfelelően a referenciapontját is. A másik épület kitörlése után csak az egyesített épület marad a listában. Ha a csoportokban gyűjtjük a módosítási előzményeket, akkor az egyesítés előtti állapot visszaállítható.

Kiértékelés, posztprocesszálás

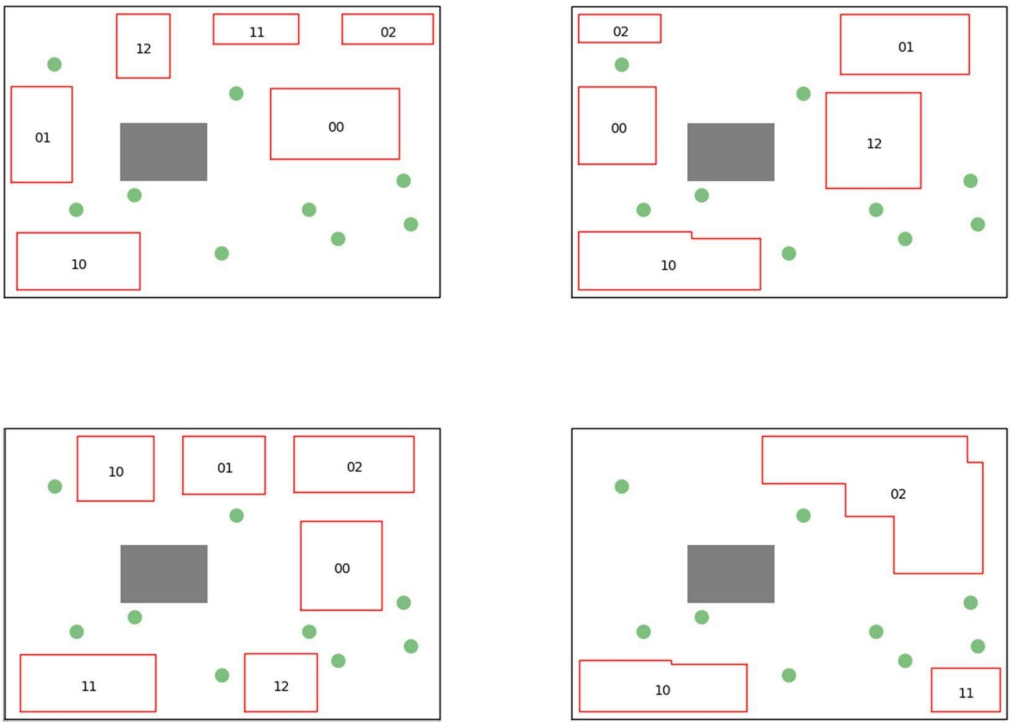
Az *Evaluator()*, vagyis a kiértékelő egy különálló osztály. Ezt jelenleg a generátor lefutása után hívjuk meg, és a benne lévő szempontok szerint értékeli a generátor eseteit, majd visszaadja a legjobb verziót, amit aztán kirajzolunk. Az értékelés jelenleg önálló algoritmusként kapcsolható a folyamatba azért, hogy bármilyen tetszőleges szempont szerint el tudjuk végezni. Ahogy az algoritmus legtöbb része, ez is cserélhető vagy kombinálható és így egyéni igényekre szabható. Az értékelés pontozáson alapul, és jelen állásban úgy keresünk minél jobb megoldást, hogy a sok generált esetből kiválasztjuk a legtöbb pontot elérőt. Amennyiben a jövőben a kiértékelő algoritmust átadjuk a generátornak, úgy az már a növesztés során kiértékelheti az adott geometriát és random lépések helyett a kiértékelés alapján jobb pontot elérő növesztéseket választhat és optimalizálhatja a beépítést.

A megjelenítést a *Matplotlib* modul segítségével implementáltuk. Minden kirajzolandó elemnek saját függvénye van, és megjelenésük, vagyis a vonaltípusok és színek változtathatók. Az esetek értékelése után a legjobban teljesítő eset kerül csak kirajzolásra. Minden egyéb lényeges információt kiiratunk az összes esetről, így ezek manuálisan is összehasonlíthatók.

Esettanulmány

Egy egyszerű példán keresztül szeretnénk bemutatni az algoritmusunk használatát. Mivel kevés módosítási lehetőséget adtunk meg az épületeknek, ezért a példánkban szereplő beépítendő telek egyszerűen téglalap alakú. A telekre elhelyezünk egy fix épületet, vagyis akadályt, amitől minden generált épületnek minél távolabb kell esniük. További nehezítésként fákat helyezünk el, melyektől megfelelő távolságot kell tartaniuk az épületeknek, ezzel is közelebb kerül a feladat a valósághoz. A pontozás úgy történik, hogy megmérjük minden épület távolságát az akadálytól és a legkisebb távolságok közül keressük a legnagyobb értéket, tehát az esetek közül azt választjuk ki, ahol a legkisebb mért távolság maximális. Ez a feladat eredményt adhat olyan valós esetekre, amikor valamilyen létesítménytől szeretnénk távol kerülni, vagy a kijelölt objektumok és a generált épületek közé üres területet, mondjuk közparkot szeretnénk létesíteni.

A feladatban egy 300x200 méteres téglalap alakú telekre keresünk beépítési variációkat. A telken kezdetben egy 30x20 méteres téglalap alakú épület helyezkedik el és 9 db 5m sugarú körrel modellezett fa. A telektől vett minimális távolság *dp*=5m, a fáktól minden épület oldala legalább 10m-re helyezkedik el. A 7. ábrán különböző variációkat láthatunk arra az esetekre, amikor minden épület egymástól legalább *db*=20m távolságra helyezkedik el (7. ábra bal oszlop) és amikor megengedett az azonos csoportban lévő épületek összeolvadása (7. ábra jobb oszlop).



(7. ábra) ~100 generált esetből 10 darab legmagasabb pontszámot elért variáció lett kiválasztva az alapján, hogy a generált épületek minél távolabb kerüljenek a fix épülettől. A 10 esetből manuális bírálat után ezt a négy esetet választottam bemutatásra.

Összefoglalás, kitekintés

A dolgozatban bemutatott evolúciós eljárás segítségével a felhasználó által megadott telekre lehet beépítési koncepciókat generálni különböző megkötések és célok figyelembevételével. A módszer segítségével a tervezési folyamat jelentősen meggyorsítható, a geometriailag lehetséges megoldások halmazának feltérképezéséhez használt kézi iterációs módszer kiváltható. A tervezés során a bemeneti paraméterek változtatásával és újabb koncepciók

generálásával alakulhat ki a végső koncepció. Az algoritmus előnye, hogy a logikája miatt cserélhetők az elemek és működése hasonló a kézi tervezéshez. Az objektumok kommunikálnak egymással, így egymástól függően választanak módosítási lépéseket. Tetszőleges értékelési, szimulációs könyvtárakat kapcsolhatunk hozzá, melyek beillesztésével bővíthetjük az optimalizálás szempontrendszerét. A módosítási lépések megválasztásáért felelős algoritmus is cserélhető, így lehetőség nyílik további fejlesztés során akár megerősített tanulás algoritmust alkalmazni.

A több szempont szerint való értékelés egy lényeges távlati célja ennek az algoritmusnak. Ha minden szempont szerint külön értékelünk és pontozunk, akkor akár előre megadott pontértékekre optimalizálhatunk beépítéséket. A több célra való optimalizálás nem végezhető hatékonyan, ha az algoritmus a módosításokat véletlenszerű próbálkozással, random változtatgatással végzi. Bevezethetünk például egy megerősített tanuláson alapuló rendszert. Ez minden eredeti funkció megtartása mellett lehetséges, elég csak a generátort kiegészíteni a szükséges funkciókkal. Adott a környezet, az eset, aminek előállnak a potenciális lépései és jutalomként szerepel a pont, amit az iterációk végén megkap az eset. Ennek többféle megközelítése is lehetséges, de ez további kutatást igényel és kidolgozása nem volt célja ennek a tanulmánynak.

Az algoritmus felépítése lehetőséget ad mélyebb építészeti összefüggések integrálására, amik belső megkötéseket adhatnak a geometriai módosításokra. További kutatások során integrálni lehet a téri elrendezésre irányuló elemzési eljárásokat, mint a space synthax [12] és gráf alapú térelemzések [19]. Ezek segítségével a használati mintázatok a geometriák létrehozására és módosítására is hatással lehetnek. Több kutatás foglalkozik azzal, hogy olyan mérőszámokat rendeljen a városokat alkotó épületek, utak, terek összességéhez, amelyek valamilyen szempontból a felhasználók jóllétét (pl. mentális egészség), komfortját kvantifikálják [16][17][18]. Az algoritmus lehetőséget ad arra, hogy az ilyen jellegű kutatások eredményeit, mint kiértékelő eljárásokat kapcsoljunk be a generálás folyamatába.

A környezetről való információgyűjtést ki lehet terjeszteni akár az épületeken belüli funkcionális viszonyok feltérképezésére, ami alapján előállnak szükséges méretek és becsülhető lesz a hasznosítható alapterület, ami a beruházás értékbecslésénél szükséges. Ezt akár egy gráfban lehet modellezni, amit a külső és a szintek közötti használati, funkcionális kapcsolatokkal is ki lehet bővíteni. Ez alapján akár épületenként előre becsülhetünk szükségesen elérendő alapterületeket, formai kötöttségeket, ami meghatározza majd egy

épületnek előálló növekedési lehetőségeit. Ha a gráfot bővítjük az objektumaink adatpontjaival, akkor egy komplex tudástár állhat össze, amiben az adott feladatot jellemző információk és kapcsolataik mind előállnak. Az objektumok innen nyerhetik az információkat és az optimális elrendezés meghatározásánál is segíthetnek.

Irodalomjegyzék

[1] Li, Peiyuan & Xu, Tianfang & Wei, Shiqi & Wang, Zhi-Hua. (2022). Multi-objective optimization of urban environmental system design using machine learning. Computers, Environment and Urban Systems. 94. 101796. 10.1016/j.compenvurbsys.2022.101796.

[2] Boyukliyski, Stoyan & Petrova-Antonova, Dessislava & Hristov, Emil & Hristov, Kristiyan. (2022). Multi-Objective Optimisation of Urban Design Using a Genetic Algorithm. 345-350. 10.1109/ICAI55857.2022.9959987.

[3] https://www.dezeen.com/2020/10/20/delve-sidewalk-labs-machine-learning-tool-cities/

[4] Krawczyk, Robert. (2002). Experiments in Architectural Form Generation Using Cellular Automata.

[5] Miao, Yufan & Koenig, Reinhard & Knecht, Katja. (2020). The Development of Optimization Methods in Generative Urban Design: A Review.

[6] Yang, L.; Li, J.; Chang, H.-T.; Zhao, Z.; Ma, H.; Zhou, L. A Generative Urban Space Design Method Based on Shape Grammar and Urban Induction Patterns. Land 2023, 12, 1167. https://doi.org/10.3390/land12061167

[7] Zhang, Jingyu & Liu, Nianxiong & Wang, Shanshan. (2021). Generative Design and Performance Optimization of Residential Buildings based on Parametric Algorithm. Energy and Buildings. 244. 111033. 10.1016/j.enbuild.2021.111033.

[8] Wilson, L., Danforth, J., Davila, C. C., & Harvey, D. (2019). How to generate a thousand master plans: A framework for computational urban design. SimAUD 2019, 113-119.

[9] Lea, Doug. (1994). Christopher Alexander: An Introduction for Object-Oriented Designers. ACM SIGSOFT Software Engineering Notes. 19. 39-46. 10.1145/181610.181617.

[10] Alexander, C. (1977). A pattern language: towns, buildings, construction. Oxford university press.

[11] Alexander, C. (1964). Notes on the Synthesis of Form (Vol. 5). Harvard University Press.

[12] Alalouch, Chaham & Al-Hajri, Sara & Naser, Abeer & Al-Hinai, Asma. (2019). The Impact of Space Syntax Spatial Attributes on Urban Land Use in Muscat: Implications for Urban Sustainability. Sustainable Cities and Society. 46. 10.1016/j.scs.2019.01.002.

[13] Ye, Xinyue & Du, Jiaxin & Ye, Yu. (2021). MasterplanGAN: Facilitating the smart rendering of urban master plans via generative adversarial networks. Environment and Planning B: Urban Analytics and City Science. 49. 239980832110235. 10.1177/23998083211023516.

[14] UrbanXTools. https://github.com/CAUPDxUrbanXLab/UrbanXTools

[15] Revit Generative Design. https://www.autodesk.com/solutions/generative-design/architecture-engineering-construction

[16] Hematian, H., & Ranjbar, E. (2022). Evaluating urban public spaces from mental health point of view: Comparing pedestrian and car-dominated streets. Journal of Transport & Health, 27, 101532.

[17] Zamanifard, H., Alizadeh, T., Bosman, C., & Coiacetto, E. (2019). Measuring experiential qualities of urban public spaces: users' perspective. Journal of Urban Design, 24(3), 340-364.

[18] Mehta, V. (2014). Evaluating public space. Journal of Urban design, 19(1), 53-88.

[19] Xie, X., & Ding, W. (2023). An interactive approach for generating spatial architecture layout based on graph theory. Frontiers of Architectural Research, 12(4), 630-650.