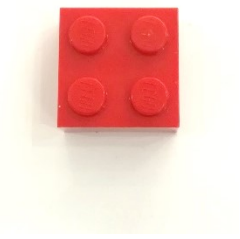
Fizikai modellen alapuló interaktív adatszimuláció városstervezési feladatokhoz

A dolgozat a Tudományos Diákköri Konferencia keretében készült.

Köszönetnyilvánítás

Szeretném megköszönni konzulensemnek, Kádár Bálintnak, hogy segített a számomra megfelelő téma megtalálásában, és véghezvitelében. A Kortárs Építészeti Központ munkatársainak is szeretném megköszönni a segítséget, és hogy lehetőséget biztosítottak számomra az interaktív modell fejlesztésében.



szerző: Dongó Tamás

konzulens: Kádár Bálint PhD

2018.

MŰEGYETEM 1782

Absztrakt

A várostervezés nem gyerekjáték. Hosszú távra előre kell gondolkodni, rengeteg adattal kell dolgozni, hatalmas területeket kell átlátni. Akárhogy is nézzük, erre csak egy komoly szakértelemmel rendelkező ember képes. Hogy lehetne mégis mindenkiből szakembert faragni percek alatt?

Erre a kérdésre keresem a választ a dolgozatomban, amely egy olyan eszköz megalkotását tűzte ki célul, melynek segítségével összetettebb problémák is egyszerűen átláthatóvá válnak bárki számára.

A rendszer alapját az MIT által létrehozott CityScope platform képezi, valamint egyéb adatvizualizációs projektekből is merítettem ötletet. A CityScope egy teljesen szabad felhasználású, várostervezést segítő eszköz, ami legokat használ a városi funkciók és kapcsolatok modellezéséhez.

Hogy az eszközömmel valós problémára lehessen választ keresni, a Kortárs Építészeti Központtal együttműködésben, az általuk a Csepel Művek területén végzett kétéves kutatómunka eredményeit használja fel. A Művek területe, Budapest legjelentősebb elfelejtett területe. Sokszáz cég működik itt, jobbára egymástól függetlenül, hol legálisan hol illegálisan. Az összegyűjtött adatok segítségével egy összetett képet kaptunk a területről, és létrehoztunk egy adatbázist.

Az eszköz három fő részből áll:

- a modell – fehér lego elemekből felépített Csepel Művek területe, valamint további lego elemek eltérő színekben
- az érzékelő – a modellt fentről egy fullHD felbontású kamera figyeli
- a vizualizációs eszköz – egy projektor helyezkedik el a modell felett, amellyel bármilyen adat rávetíthető

A fejlesztés során a Processing 3, java alapú környezetet használtam, valamint az OpenCV kiegészítő könyvtárat. A szoftver egy laptopon fut, amihez a kamera csatlakozik. A kamera képének feldolgozása után, amennyiben azt érzékeli, hogy a modellben változás történt – ami az egyes épületek színesre cserélésével érhető el – átírja az adott épület funkcióját, amivel az ehhez tartozó értékek is változnak. Az adatbázisból kinyeri a környező épületek és funkciók különböző értékekeit (pl. zaj, károsanyag kibocsátás, gyalogos forgalom növelése, gépjármű forgalom növelése). Egy algoritmus ezekkel újra számítja a különböző értékeket, amikkel

folyamatosan frissíti a képet, amit a projektor a modellre vetít, ezzel szolgáltatva információt a felhasználónak arról, hogy az intervenció (az épület funkciójának megváltoztatása) hasznos, vagy épp káros volt.

Abstract

Urban planning is not child's play. One must think ahead, deal with huge data, and understand the complexity of a huge area. It is obvious that only an experienced expert is capable of such a task. Is there a way that anyone could be an expert in minutes?

I am looking for the answer in this thesis, which is about building a tool, that would make complex urban problems easy to understand for anyone.

The basis of the system is a platform made by MIT, called CityScope, and I also got some ideas from other data-visualization based projects. CityScope is a fully open-source urban planning tool, that uses legos to model urban functions and their connections.

To find the answer to real life scenarios, in collaboration with the Contemporary Architecture Centre, my project uses the data they collected in the past 2 years, in the Csepel Works area. This territory is Budapest's most important, forgotten area. Many hundreds of companies operate here, mostly independently from each other, sometimes legally, other times illegally. From the gathered data, we can now see a very complex image of this area, and we could create a database.

The system consists of 3 main parts:

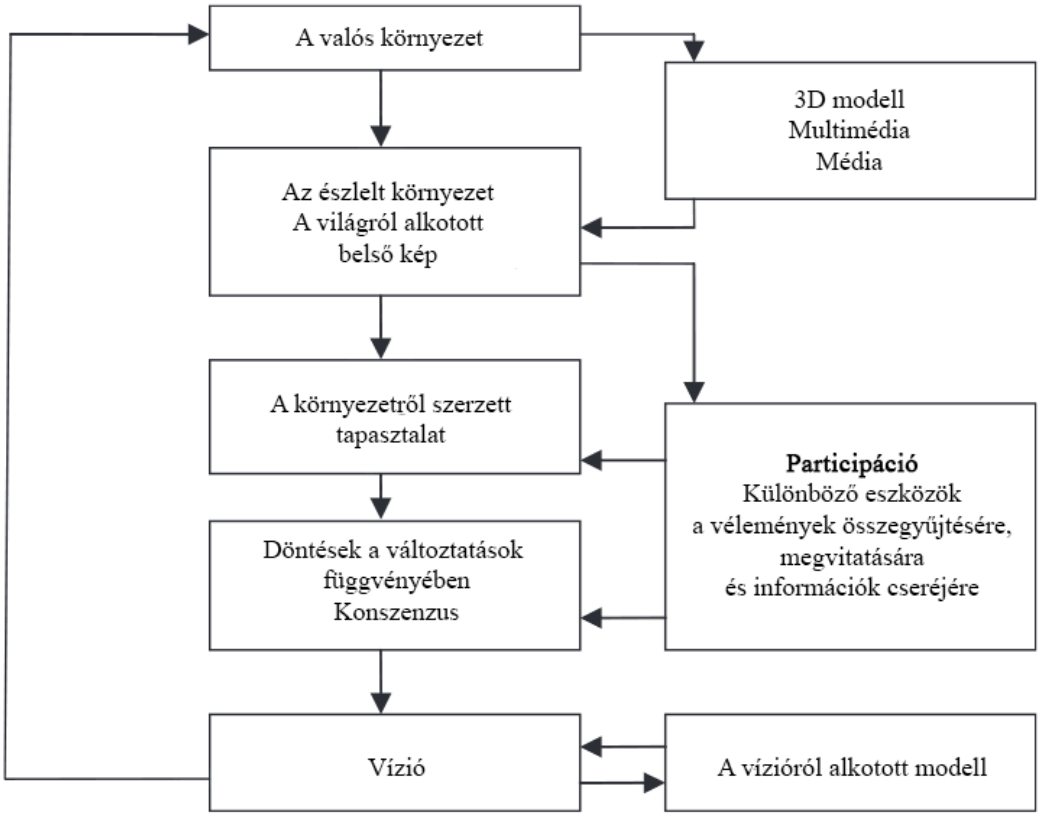
- the model – the Csepel Works model is made from white lego bricks, and there are some more bricks of different colours as well
- the sensor – the model is viewed by a fullHD resolution camera from above
- the visualisation tool – above the model, there is a projector, which can project any data on it

During development I used Processing 3, a java-based environment, and OpenCV as well, an extension library for the camera. The software runs on a laptop, to which a camera is connected. After processing the image of the camera, if the system thinks there was a change made in the model – which can be achieved by replacing the white bricks with coloured ones – it replaces the given building's function, which will cause its values to change. From the database, it gets all the data about all the nearby buildings and functions (e.g. noise, pollution, increasing the pedestrian or vehicle traffic). An algorithm constantly recalculates the different values, and using them creates a picture, which is projected on the model, that provides information to the user, whether their intervention (changing the function of a building) was useful, or harmful.

Tartalomjegyzék

Bevezetés

A kortárs várostervezés elmélete és gyakorlata egyre inkább a kommunikációra és a vitákra helyezi a hangsúlyt. Célja egy egységes vízió megalkotása, amiben a helyi lakosság szava döntő szereppel bír a városi terek kialakítása során. A participáció során, amely a lakosság bevonásával történő döntéshozatalról szól, több lépésen keresztül próbálnak eljutni az optimális, legtöbbek számára kielégítő eredményhez. Ez a hosszú folyamat sok tervezést igényel, a következő ábra jól szemlélteti az összetettségét.



A participáció elhelyezkedése egy vízió kialakításának folyamatában. (Hanzl, 2007)

Ahogy az ábrán is látható, a folyamat egy ciklusként újra és újra kezdődik. Attól függően, hogy kik a résztvevői a folyamatnak más és más érdekek miatt, más és más víziókat kapunk eredményként. Ezek összehangolása már a várostervezők feladata, amely általában sok fejtörést okoz nekik.

A jelenleg is használatban lévő „whiteboard-os" vagy „cetlis" ötletelés már idejét múltnak számít. Ilyen workshopokra általában kis létszámmal jelennek meg az érdekeltek, talán azért, mert nem tudják, hogy van lehetőségük aktívan változtatni környezetükön, vagy mert úgy érzik nem értenek a várostervezéshez. Így releváns információ nehezen gyűjthető, az így kialakult

vízió általában elnagyolt, nem feltétlenül lesz mindenki számára elfogadható. Úgy gondolom, az új technológiák bevonásával növelhető a részvételi arány, amivel mindenki számára élhetőbb környezetet tudunk létrehozni, amit a sajátjuknak érezhetnek.

A dolgozatom a következő kérdésre próbál választ adni: *Hogyan lehetne a modern technológia segítségével egy olyan közös platformot biztosítani a témában érdekelt lakosság, önkormányzat és cégek számára – legyen szó a legkisebb vállalkozástól a legnagyobb nemzetközi vállalatig – amely közös fórumot biztosít számukra az érdekeik és elképzeléseik megvitatásához, és egy közös, mindegyik fél számára elfogadható vízió kialakításához?*

A város az emberek és az épített terek kapcsolatán alapszik. A városi ökoszisztéma koncepció a vizsgált várost vagy a városrészt egy önálló rendszerként képzeli el. Azon alapszik, hogy egy ökoszisztéma lehet kicsi vagy nagy. Ha egy erdő közepén egy rothadó fatörzs élővilágát vizsgáljuk rendszert vélhetünk felfedezni, de ha két különböző élővilágot egyesítünk is ökoszisztémát alkotnak, például egy hegyi forrás és egy sivatagi cserjés organizmusainak és fizikai környezetének az együttese is képezhez egy rendszert. (Pickett, és mtsai., 1997)

Ennek modellezése nehéz feladat, teljes részletezettségű képet nem is kaphatunk, több esetben közelítéseket kell alkalmaznunk. Függ attól, hogy mennyi és milyen típusú információ áll rendelkezésünkre a vizsgált területről.

Ahhoz, hogy egy jól használható participációs platformot hozzunk létre, először egy összetett, részletes és rendszerezett képet kell alkotnunk a vizsgált környezetről. Már meglévő adatbázisok felhasználásával és helyszíni mérésekkel is gyűjthető adat. Ezek adatbázisban kerülnek tárolásra, amelyből a későbbiekben kinyerhetjük a számunkra szükséges adatokat.

Ezt az információhalmazt kézzelfoghatóvá kell tenni a participációban résztvevők számára, hogy ők is hatékonyan tudjanak vele dolgozni. A következő fejezet különböző példákon mutatja be hogyan lehet megismertetni a lakossággal összetett koncepciókat, milyen platformokon keresztül teszik ezt, és milyen eredménnyel tudják ezt véghez vinni.

Megvalósult participációs platformok

Hagyományos részvételi tervezés

A participáció megvalósításához sokféle tevékenység összehangolására van szükség. Ide tartoznak az általános tréningek, amellyel a csoport együttműködése fejleszthető; a közösségi találkozók, amelyen moderátorok segítségével a közös ötletelés, döntéshozatal, stratégiai tervezés kerül előtérbe.

Egy hagyományos participációs esemény több részre bontható. Egy bevezető előadással kezdődik, amely során tömören és lényegre törően összefoglalják azokat a keretinformációkat, amelyek feltétlenül szükségesek a további munkához. A második rész – a műhelymunka – lebonyolítására, a résztvevők létszámától függően, többféle lehetőség adódik. Kártyatechnika, elmetérképezés, szavazás, kávéház módszer mind különböző technikák, amelyekkel ötletbörze lefolytatható.

Közös mindegyik technikában a moderátor jelenléte. Mederben tartja a hozzászólásokat, konstruktív légkört teremt, fókuszálja a beszélgetést, időnként összegzi az addig elhangzottakat.

Nagyon fontos, hogy minden műhelymunka egy kérdésfeltevéssel kezdődik. Le kell szögezni, hogy mi az, amire választ vagy megoldást kívánnak kapni az esemény végén.

Ezután ötletbörze következik, amely során mindenki egy cetlire leírja az ötleteit, véleményeit, pár szóban, röviden. Ezt követően mindenki felolvassa amit írt, majd kiragasztják a cetliket mindenki által látható helyre (whiteboard, csomagolópapír, tábla).

A gondolatok ismeretében következhet az elemzés, a cetliket, a probléma jellegétől függően, csoportosíthatják, sorba állíthatják vagy szavazhatnak róluk.

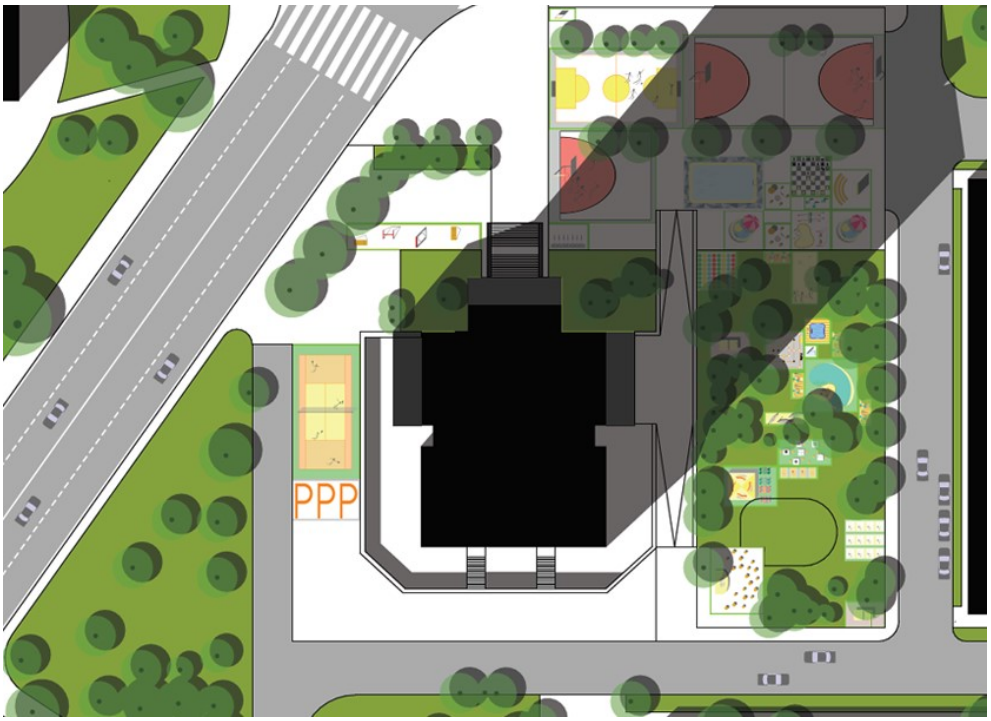
A műhelymunka végén, minden résztvevő elmondja, hogy mit jelentett számára a közös munka, teljesültek-e az elvárásaik, hogyan érezték magukat.

A találkozó után az ötleteket dokumentálják, esetekben elérhetővé teszik online felületeken keresztül, így azokhoz is eljut, akik nem tudtak részt venni az eseményen. A kiértékelésről, és az ötletek valóságba ültetéséről már a városfejlesztésért felelős szervezetek gondoskodnak. (Sain & Rab, Részvételi tervezés a településfejlesztésben és településrendezésben, 2018)

SCHURB

A Budapesti Műszaki Egyetem Urbanisztika Tanszék 2011-ben, a Schönherz kollégium lakóinak bevonásával a kollégium közvetlen környezetének áttervezését tűzte ki célul. Ehhez

építész és informatikus hallgatókkal egy online játékot terveztek. A városi téralakítási tapasztalatokat, valamint a létező virtuális térszimulációs játékokat egyesítve hozták létre a schurb.org című internetes honlapot. Ezen egy játék futott, aminek a pontos szabályrendszerét is a hallgatók találták ki.



Egyik a schurb.org internetes holnapon keresztül, a kollégium lakói által készített tervek közül

Röviden a játék menetéről: klubokat lehet alapítani, amelyekbe a regisztrált lakók léphetnek be. Amennyiben a klub létszáma elérte a megfelelő létszámot, lehelyezhették a kívánt funkciójukat a térképre. A klubokon belül és között is volt lehetőség üzenni, így a személyes meggyőzés továbbra is fontos paraméter maradt a játék során.

Remek lehetőség volt ez a kollégisták számára, hogy felismerjék a lehetőségeket az épített környezet formálásában. A platform megválasztása is kedvező volt számukra: minden hallgató rendelkezett internethozzáféréssel, és játékosan, egymással versengve tudtak új víziókat létrehozni.

UN Habitat – Block by Block

Kirtipur egy ősi város kb. 5 km-re Nepál fővárosától Katmandutól. Az ENSZ támogatásával a helyi civil szervezetek, és közösségek megpróbálták a köztereket megőrizve újragondolni azokat, hogy élhetőbbek legyenek a helyiek számára. A várostervezés hiánya és az illegális beépítések miatt, a minőségi terek aránya gyorsan csökken a városban. Hogy ezt a tendenciát

visszafordítsák, az ENSZ szakértői a Minecraft nevű számítógépes játékkal próbáltak minél több embert bevonni a tervezésbe. (von Heland, Westerberg, & Nyberg, 2015)

A Minecraft a világ egyik legnépszerűbb számítógépes játéka. A felhasználó szabadon változtathatja a teret, a blokkok lehelyezésével vagy összetörésével. Ahhoz, hogy a workshopokon a városi kérdésekre tudjanak megoldást keresni, először megmodellezték a tereket tervek és képek alapján, majd kisebb csoportokat kialakítva külön próbálták jobbá tenni azokat.

A vizsgált környezetről készült fotó és a Minecraftban megépült mása, már a fejlesztéseket is tartalmazza (von Heland, Westerberg, & Nyberg, 2015)

teamLab – Block Town

A legnagyobb hatással a japán kollaboráció, a teamLab, Block Town című projektje volt rám. A teamLab egy kreatív csapat, rengeteg vizuális és audiovizuális projekttel foglalkoztak, hatalmas sikerrel világszerte. 2018-ban hazánkba is ellátogattak a Future Park nevű

kiállításukkal. A Párizsi udvar falai közt nyílt meg, ahol megannyi tudományos és a kreatív gondolkodást fejlesztő, interaktív játékkal lehetett megismerkedni.

A technológiát tekintve, valamennyi játéknak közös ismertetőjegye, hogy pusztán projektorokat és Kinect-eket – a Microsoft által az Xbox konzolcsaládhoz tervezett mozgásérzékelő kamerákat – használnak. Az interakciót a látogatók mozgása, az általuk megérintett felületek, vagy egyes tárgyak áthelyezése képezik.

A Kinect érzékelőkben kétfajta kamera helyezkedik el, egy hagyományos, és egy infravörös kamera. A hagyományos kamera a színek érzékeléséért felel. Az infravörös kamera, egy infravörös lézer projektor képét rögzíti, amely apró pontokat, meghatározott mintában rávetíti a vizsgálandó felületre. Ezekről a pontokról készít az infravörös kamera egy képet, melyet a beépített mikrovezérlő mélység adattá konvertál, azaz megadja az egyes pixelek távolságát a kamerától.

Gyermekek játszanak a teamLab Block Town című installációjával – forrás: teamlab.art

A Block Town szintén a projektor és a Kinect kamera párosát használja. Az asztalon elhelyezett testek – kúp, ház forma, négyzet alapú hasáb vagy felhő forma – érzékelése fentről történik, a kamera az asztalt fentről nézi. A testek színezettek, a kúp kék, a vizet szimbolizálja, a ház tetője zöld, a négyzet alapú hasáb sárga, és helikopter leszállót készíthetünk vele, a felhő forma fehér és repülők útvonalát mutatja.

Sajnos nem találtam róla semmilyen információt, de úgy gondolom, hogy a Kinect mélység érzékelőjével tud a rendszer információt szerezni a tárgyak elhelyezkedéséről. Elszeleteli több síkkal az asztal feletti teret, és megnézi, hogy az adott síkokban milyen formák jelennek meg,

pl. a ház forma esetében egy magasabb síkban, ahol a tetőt metszi el, egy kisebb téglalapot, lejjebb, egy nagyobbat érzékel, a kúp esetében egy kisebb majd egy nagyobb átmérőjű kört.

Miután felismerte az alakzatokat egy algoritmus készít egy animációt, pl. a két házat összeköti egy úttal, amin autók járnak, a két kúpot egy folyóval, amin hajók járnak. Ezt az animációt egy vagy több projektor az asztalra vetíti.

MIT – CityScope

A CityScope elnevezés nem egy eszközt jelöl, hanem inkább egy technológiát. Segítségével a felhasználók módosíthatják a vizsgált területről készült fizikai modellt, és valós időben kaphatnak visszajelzést az intervenciók hatásáról. Azzal, hogy valós idejű a visszacsatolás, a közös megegyezés és a közös cél elérése sokkal gyorsabbá válik. (Alonso, és mtsai., 2018)

Különféle szcenáriókban és környezetekben használták az elmúlt évek során, minden esetben az adott szituációra szabva magát az eszközt is. Egyik leghíresebb a hamburgi CityScope története. 2015-ben a nagy számmal Hamburgba érkező menekültek miatt, a polgármester felkereste a CityScope feltalálóit. A problémát a menekültek elhelyezése jelentette. A helyiek bevonásával, 2 hónap alatt 150 telket azonosítottak, melyek potenciálisan alkalmasak lehettek a menekültek befogadására, vagy új épület építésével, vagy a meglévő felújításával. Ezzel 1500 menekültnek tudtak szállást biztosítani, és ez a szám a következő időszakban tovább nőtt. (Noyman, Holtz, Kröger, Noennig, & Larson, 2017)

CityScope Andorra – mozgatható elemek képezik az interakciót – forrás: media.mit.edu

A másik kiemelkedő hasznosítása a platformnak, Andorra városához köthető. A városvezetés kíváncsi volt a helyiek és a turisták viselkedési szokásaira, a nagyobb rendezvények, mint a Cirque de Soleil vagy a Tour de France alatt. Ehhez rendelkezésükre állt a város területén használt telefonok geopozíciós adatai (térbeli helyzetük és sebességük). Ennek segítségével tudták modellezni, hogy melyik utakat preferálják a helyiek és a turisták a közlekedéshez. (Grignard, és mtsai., 2018)

Ami minden CityScope projektben megegyezik az a LEGO. Minden fizikai modellt LEGO építőelemekből készítenek el. Ahhoz, hogy könnyen módosítható maradjon, raszterre bontják a modellt, amelyben a kisebb elemeket kicserélve tudnak beavatkozásokat szimulálni. Amit még nagyon fontos megemlíteni a technológiával kapcsolatban, hogy az eredmények teljesen szabad felhasználásúak, az elkészült szoftverek nyílt forráskódúak.

Minden felhasználásában közös, hogy a LEGO elemek egy olyan asztalon helyezkednek el, aminek a felső lapja átlátszó, plexi. Erre azért van szükség, mert az egyes épületek, így a LEGO-k tulajdonságait a legalsó rétegük színes elemekkel való helyettesítésével változtathatjuk. Ezt a változást az asztalba épített kamera regisztrálja, ami egy tükör segítségével látja be az asztal teljes felső részét. Fontos még, hogy az asztal belseje egy zárt rendszert képez, saját világítással rendelkezik, és minden oldalról zárt, így a külső hatásoktól – mint a fényviszonyok változása – teljesen mentes. Ennek megfelelően ezek az asztalok teljesen egyedi gyártásúak, a terveik szintén elérhetők online. A kamera képét egy megfelelő teljesítményű számítógép feldolgozza, majd az előre megírt szoftver, algoritmusok alapján kiszámolja a képet, amit egy projektor fentről rávetít a makettre.

A rendszer a LEGO elemek pontos és átgondolt tervezésén és gyártásán alapul. Az elemek modulárisak, tehát adott raszterben többféle képen összeépíthetők. Egy LEGO „egység”, ami tulajdonképpen a LEGO raszterének mérete, 8 mm. Ahhoz, hogy ez az egység állandó maradjon, nagyon pontos megmunkálást használnak a gyárakban: fröccsöntést. A kockák alapanyag az ABS (akrilnitril-butadién-sztirol), ami egy nagy keménységű és szilárdságú, jó hő és vegyszerállóságú polimer. Ezt az anyagot, a fröccsöntő gép magas hőmérsékleten előre elkészített formákba préseli, melyek nagy pontossággal, kis felületi érdességgel tartalmazzák a LEGO elemek negatívjait. Több érzékelő is helyet foglal az öntőformákban, melyek folyamatosan ellenőrzik a hőmérséklet és nyomásváltozásokat, ezzel is növelve a precíz megmunkálást. A gyártósor olyan finoman van hangolva, hogy (a LEGO adatai szerint) minden egymillió kockából csak 18 bizonyul selejtnek. Ezzel a technikával képesek voltak létrehozni egy olyan egységes rendszert, amely a világ egészén ismert.

Az MIT folyamatosan dolgozik azon, hogy minél széleskörűbben megismerjék a technológiát, és a világ több pontján is használják.

Kollaboráció a Kortárs Építészeti Központtal

A Kortárs Építészeti Központ (röviden KÉK) egy független szakmai szervezet, amely az építészet oktatását, megismerését és fejlesztését kívánja előremozdítani a lehető legszélesebb szakmai és szakmán kívüli környezetben. Számos kiállítás, rendezvény és program fűződik a nevükhöz: Budapest100 fesztivál, Budapesti Építészeti Filmnapok, közösségi kertépítések, olyan események, melyekkel próbálják a várost és a városban élőket közelebb hozni.

A IV. Nemzetközi Építészeti Makettfesztivál szervezésében is részt vállal a KÉK. A fesztivál 2019 tavaszán kerül megrendezésre, és a makettek és a digitális adatok kapcsolatával foglalkozik majd. Ennek előzeteseként kívántak létrehozni egy kiállítást, amelyen bemutathatnak egy olyan interaktív modellt, amely a CityScope-hoz hasonló technológián alapul, és akár várostervezési feladatok ellátására is alkalmas lehet. A továbbiakban arról lesz szó a dolgozatomban, hogy a kezdeti ötlettől hogyan jutottunk el a kiállításig, amin bemutatásra került a modell.

Shared Cities

A Shared Cities: Creative Momentum (röviden SCCM) egy európai kulturális platform, melynek küldetése az európai városok élhetőségének a javítása. 6 ország 11 szervezetét hozza össze, hogy teret biztosítsanak az építészetnek, a művészeteknek, és hogy együtt hozzanak változást a városi életben. A KÉK a Budapest100 fesztivállal kapcsolódott be a Shared Cities programjai közé, majd 2016-ban kezdték meg a kutatómunkát a Csepel Művekkel területén. A kutatás 2 éves eredményét foglalta össze az a kiállítás amire az interaktív installáció is készült.

Több workshoppal és prezentációval szeretnék feltárni és megvitatni a lehetőségeket, melyeket ez a hányattatott sorsú terület hordoz magában.

Csepel Művek

Csepeli Vas- és Fémművek északi része. (1963) – forrás: fortepan.hu

A Csepel Művek a Csepel-sziget északi részén, több mint kétszáz hektáros területen elhelyezkedő régi iparterület. Jelenleg Budapest egyik legfontosabb és az ország legnagyobb egybefüggő barnamezős zónájaként szerepel a térképeken. Az 1886-ban alapított Weiss Manfréd Lőszergyár az évek során négy jelentős változáson ment keresztül, ezek nyomai még ma is felismerhetők az épületeken, az utakon, az infrastruktúrán, a vállalati szerkezetén és a bent dolgozókon. A legnagyobb változást a Művek struktúrájában a rendszerváltozás után bekövetkező hirtelen és koncepció nélküli privatizáció, melynek hatására a terület atomjaira bomlott. Az elmúlt évtizedekben sok, az eredeti ipari tevékenységtől eltérő profilú cég költözött be, ezzel táptalajt biztosítva egyéb cégek betelepülésének is. Az ad-hoc, koncepciómentes átalakulás miatt az infrastruktúra és az épületek állapota is romlásnak indult. A terület a cégek olvasztótégelyévé vált.

A Művek területén jelenleg több mint 400 tulajdonos és több száz bérlő van jelen. Az ingatlanok többsége osztatlan közös tulajdon, állami tulajdonba csak a fontosabb utak és közművek tartoznak. Az összetett tulajdonosi rendszer miatt az egységes jövőkép kialakítása komplex feladat.

Ennek az összetett problémának a megoldására, csak az érdekelt felek bevonásával lehet rátalálni. Mindannyian érdekeltek abban hogy cégük jövője hogyan fog alakulni, így különböző workshopokon keresztül lehetőség lenne egy közös vízió kialakítására. A telkek tulajdonjogainak, az utak állapotának alakulása, a cégek fejlődésének mértéke, a területhasználat változása mind olyan faktor, melyek különböző eseteire, különböző jövőképet lehetne szimulálni. Erre lenne alkalmas egy interaktív adatvizualizációs eszköz, amely kézzelfoghatóvá tenné a várostervezést.

Azt a célt tűztük ki magunk elé, hogy a CityScope technológiához hasonló, viszont olcsóbb és hatékonyabb eszköz létrehozásával, valamint egyedi algoritmusok és szimulációk kifejlesztésével hozunk létre egy participációs platformot, melynek segítségével a Csepel Művek érdekelti köre képes lesz egy közös vízió kialakítására.

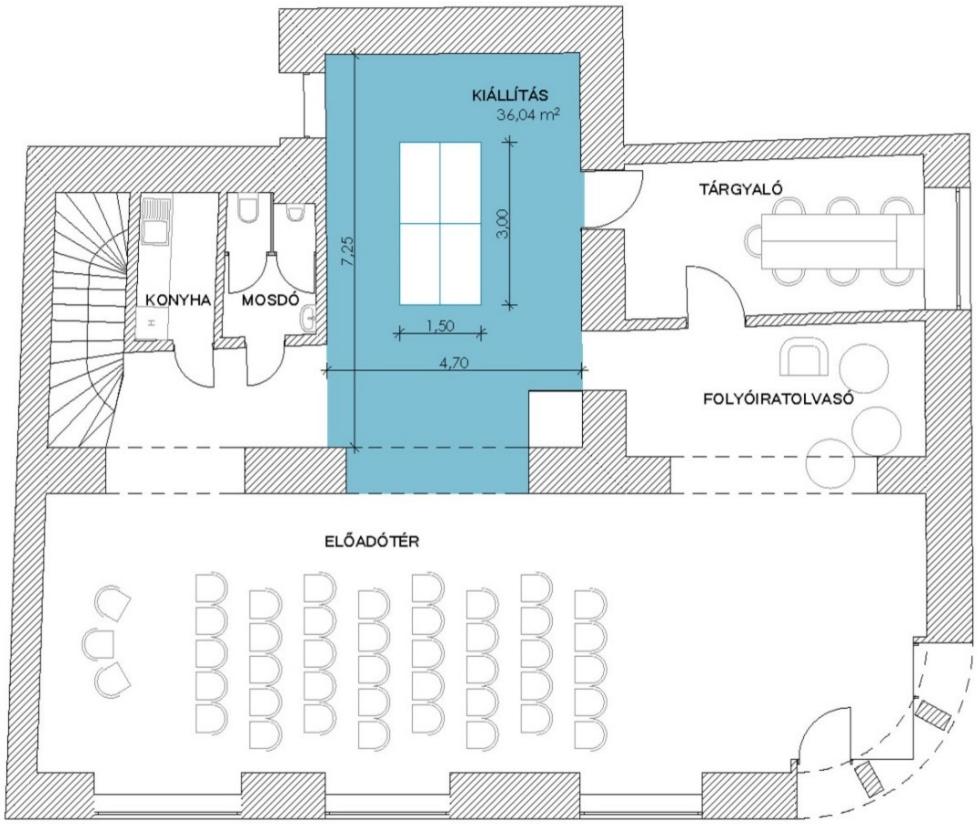
Átültetés a valóságba

Előkészítés

Először az installáció méreteit határoztuk meg. A kiállítóterem adottságait (alapterület: 7,25 m × 4,70 m, belmagasság: 3,80 m) figyelembe véve, az 1:1000 léptéket határoztuk meg a makett végső méreteinek, ezzel kb. 3 m × 1,5 m nagyságúra becsültük a befoglaló méreteit. Úgy gondoltuk, hogy ekkora léptékben még nem szükséges nagymértékű részletezettség, és nem is olyan aprók az egyes épületek, hogy kézzel nehézséget okozzon a mozgatásuk. Ezzel a méretaránnyal, a LEGO rasztere, ami a modellen 8 mm, a valóságban 8 méternek felel meg. Tehát egy egységnyi építőelem egy kisebb lakóépületet szimbolizálhat, két egységnyi egy 16 méter széles épületet, pl. egy irodaépületet, négy egységnyi, ami 32 méter akár egy csarnoképületet is.

A kiállítóterem belmagassága a projektor kiválasztásában került előtérbe. Nyilván való volt, hogy nagy asztalt csak egy nagyobb látószöggel rendelkező projektor képes teljes egészében bevilágítani. A modell földszinttől mért magasságát 80 cm-ben állapítottam meg, így elméletileg egy 3,00 méteres távolságon kellett egy legalább 3,00 méter széles képet vetítenünk. Ezt a tulajdonságát a projektornak a vetítési arányként definiálják a gyártók, ami a vetítési

távolság és a vetített kép szélességének hányadosa. Esetünkben ez pontosan 1. Ennek megfelelően 1 vagy annál kisebb értékkel kerestünk projektorokat, végül, egy 1,15-ös vetítési arányú projektort sikerült beszerezni, és ezzel kellett megoldanunk, hogy az asztal egésze vetíthető legyen. A választott projektor: *BenQ MH760 lumen Full HD üzleti projektor*.



A KÉK alaprajza, kék színnel jelölve a kiállítótermet, benne a makett asztalának hozzávetőleges pozíciójával és méreteivel.

A kamerát hasonló módon kerestem, igaz ebben az esetben könnyebb volt megfelelő terméket találni, mivel a legtöbb webkamera lászöge meghaladja a szükséges 53,13°-os szöget. Fontos szempont volt még a nagy felbontás, és az alacsony árkategória, így egy full HD (1920×1080 pixel) felbontású kamerára esett a választásunk. A választott webkamera 78°-os látószögű: *Logitech 920 HD Pro.* További előnyt jelentett ezzel a típussal kapcsolatban a szoftveresen szabályozható fókusztávolság és fehéregyensúly.

A központ ezen kívül még a rendelkezésemre bocsátott egy 2007-es iMac kompakt asztali számítógépet. A macOS, az operációs rendszere nagyon stabilan és megbízhatóan működik a Machintos számítógépeknek, viszont, ezen típus alacsony teljesítménye miatt, más alternatívát kellett keresnünk. Egy ritkábban használt KÉK-es laptopot is megkaptam, így végül azt tudtuk használni. Igaz, hogy azon egy kevésbé stabil Windows 10-es operációs rendszer futott, viszont

a teljesítménye sokkal jobb volt, mint az iMac-nek, és a tesztek során elég stabilnak is bizonyult ahhoz, hogy egy egész napon keresztül fusson rajta a szoftver, ami a vetítést végzi.

Szükség volt továbbá, egy projektor tartó konzolra. Erre egy univerzális projektor konzolt találtam, típusa: *Meliconi PRO 100.*

Ezen kívül tükörrel oldottam meg az alacsony belmagasságból és a projektor kis látószögéből adódó gondot, hogy a projektor nem képes bevetíteni a teljes modellt. Ehhez egy 310 mm × 430 mm méretű, 3 mm vastag tükröt használtam. Megtámasztására barkácsboltokban beszerezhető, fenyő polc tartó konzolokat találtam, amelyek egy 45°-os léccel vannak gyárilag megerősítve, így arra a részre lehetőségem volt felerősíteni a tükröt. A tükrön kívül a kamera is a tükröt tartó konzolok egyikére került felhelyezésre.

Installálás

A kiállítóterem szerelt homogén álmennyezettel rendelkezik. Egy szerelőnyílást is elhelyeztek rajta, ami megkönnyítette a projektor és a tükör felszerelését is. A tartóbordákra kereszt irányban fektetett kábeltartó vasakhoz rögzítettem a projektort és a tükröt tartó konzolokat, így nem a gipszkarton lemezeket terhelték, nem állt fent a csavarok kiszakadásának veszélye.

A projektor, a tükör és a kamera egymáshoz viszonyított helyzete és rögzítésük.

A projektor és a kamera kábeleinek hosszabbítása is szükséges volt, mivel a laptop viszonylag távol helyezkedett el a vetítéstől. Ehhez a hálózati feszültséget biztosító kábelhez egy 15 méteres hosszabbító kábelt, a kamera USB kábeléhez egy 10 méteres erősítővel ellátott USB

hosszabbítót kötöttünk. A projektorhoz kapott HDMI kábelt egy 10 méteres darabra cseréltük. A kábeleket a mennyezet síkjában majd a fal mellett vezettük el a laptopig.

Az asztal megfelelő pozícióját a projektor az asztal síkjára vetett képéhez kerestem meg, hogy a lehető legnagyobb képpel tudjak dolgozni.

A projektor beállításainál szükség volt a kép tükrözésére, mivel a tükörképét látnánk az alapbeállítások használatával. Szükséges volt még a kép torzulásainak korrekciója is, a sarokpontok mozgatásával tudtam a „trapézosságot” megszűntetni.

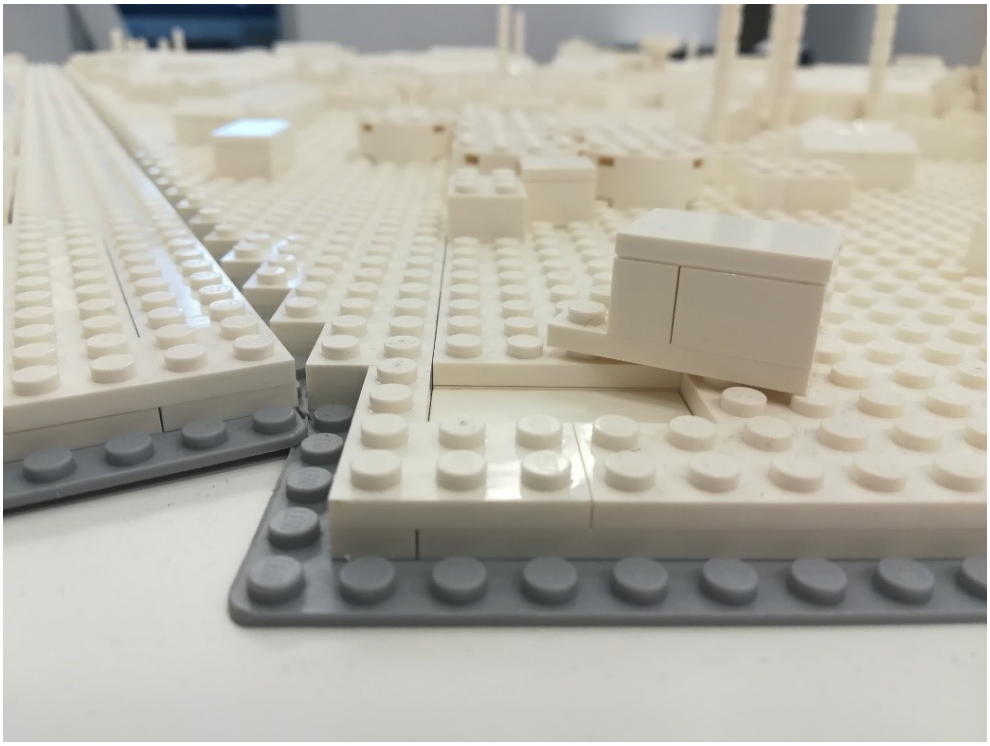
LEGO modell

A CityScope esetében az adott épület funkciója a kivehető egység alján lévő LEGO elem más színű elemre való cseréjével módosítható. Úgy gondoltuk, hogy azzal, ha a kamerát a modell fölé helyezzük, és az elemeket fentről vizsgáljuk, azzal ugyanazt az eredményt érhetjük el, amennyiben az intervenciót a felhasználó az egész épület más színűre cserélésével jelzi. Ezzel egy sokkal széleskörűbben használható rendszert kapunk, hiszen így már nincs szükség az egyedi gyártású asztalokra, bármilyen sík felületen megépíthetővé válik.

A Csepel Művekről készült légifelvétel, és az elkészült digitális modell. – forrás: mapbox.com

A Csepel Művekről készítettünk egy 3D virtuális modellt. Ebben egy, akkor a KÉK-ben gyakornokként dolgozó cseh egyetemista fiú volt a segítségemre, aki már korábban is tervezett LEGO épületeket számítógépen, valamint a Csepel Műveken végzett kutatáshoz is szorosan kapcsolódott a munkája, több felmérést végzett a területen, készített zajtérképet is, melyet később interaktívan meg is jelenítettünk a kész maketten.

A számítógépes modellezés kezdete előtt megbeszéltük a részletek, elkészítettem neki vázlatosan a rasztert, amelybe az épületeknek illeszkedniük kell (ami már tartalmazta az északi és a déli rész egymáshoz képest való elforgatását), valamint, hogy az utak elkülöníthetők legyenek, megállapodtunk, hogy a valóságban is szélesebb, főbb utak szélessége 4 egység (4 × 8 mm = 32 mm), az alárendelt utak szélessége 2 egység (2 × 8 mm = 16 mm) legyen. Ezután szabad kezet biztosítottunk neki. A tervezés során emlékeire hagyatkozva, a Google Maps nagyfelbontású 3D nézetét, valamint utcakép funkcióját használva, kevesebb mint egy hét alatt elkészült a teljes modellel. Tervező programként a Stud.io nevű szoftvert használta, amely egy nem hivatalosan LEGO által készített eszköz, viszont tartalmazza az összes valaha készült építőelemet, ezzel meggyorsítva a tervezés folyamatát. Próbálta minimalizálni az elemek sokféleségét, így végül **79 különféle elemből épül fel** a 3D modell.



A LEGO modell réteges felépítése.

Úgy terveztük meg a makettet, hogy a lehető legegyszerűbben módosítható legyen. Legalsó rétegként „baseplate” elnevezésű szürke, vékony, nagyobb alapterületű, hajlékony lemezeket

használtunk. Ez adja a modell merevségét. Erre került rá a réteg, ami biztosítja, hogy az elemek könnyen mozdíthatók legyenek, úgynevezett „tile-ok”, melyeknek a hagyományos elemekkel ellentétben, nincsenek az összekapcsolást lehetővé tevő „bütykei”. Az utaknál és a beépítésre nem szánt területeken hagyományos összekapcsolható elemek kapnak helyet, két rétegben, átlapolva, közrefogva a módosítható elemeket.

Azzal, hogy digitálisan is megépítettük, és a szoftver adottságainak köszönhetően, sok információt kaptunk a kész modellről. Ki tudtuk listázni az elemeket fajták, színek szerint, pontos elemszámot tudtunk kinyerni, pontosan meg tudtuk határozni a befoglaló méreteit, amit addig csak körülbelül tudtunk megállapítani, valamint meglepő adatokat kaptunk a modell tömegére is, több mint **20 kg**-ban állapította meg a szoftver az össztömeget.

Az egyes elemek pontos típusát és darabszámát elküldve a LEGO magyarországi képviseletének, két héten belül megérkezett a makett megépítéséhez szükséges összes elem. A KÉK munkatársai közül több mint tízen, 4 napon keresztül építették a modellt.

Így, hogy már kész volt a fizikai modell, és a minden perifériát telepítve volt, megkezdődhetett az modell interaktivitásának szoftveres építése is.

A szoftver felépítése

Processing

A szoftver megírásához és teszteléséhez a Processing 3.3.7 Java alapú fejlesztő környezetet használtam. A Processing egy könnyen használható, ingyenes letölthető, nyílt forráskódú kódszerkesztő, -fordító és -megjelenítő program. 2001-ben, kifejezetten a vizuális alkotás létrehozásának támogatására hozták létre, ezért szintaxisa a leggyakrabban használt programozási nyelvek leegyszerűsített változata, így könnyen elsajátítható bárki számára. Célközönségük a diákok, tervezők, kutatók, és bárki, akit akár hobbi szinten érdekel a programozás. (Reas & Fry, 2007)

Ahhoz, hogy a kamera által küldött képet a Processing fel tudja dolgozni, két könyvtár telepítése szükséges. Könyvtárakat a programon belül van lehetőség telepíteni, magától elvégzi a letöltést és az installálást, melynek befejeztével egyből használható is lesz. Az egyik könyvtár a Processing sajátja, Video elnevezésű. Ennek a segítségével képes lesz a program a kamera képét lekérdezni. A másik könyvtár Greg Borenstein szabad felhasználású kódját tartalmazza, és az

OpenCV for Processing 0.5.4 nevet viseli. Ez a kapott kép feldolgozását fogja elvégezni, különböző korrekciókon keresztül.

A kamera képének feldolgozása

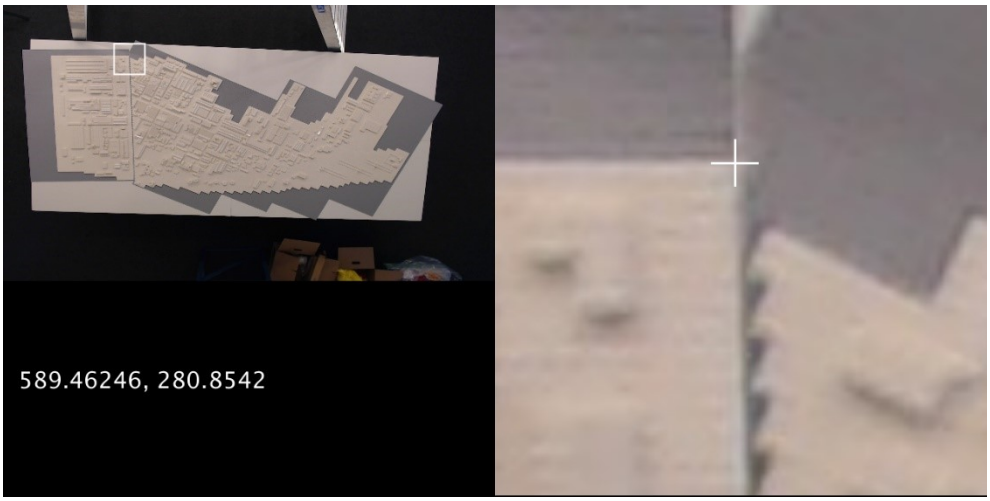
Egy sketch – ahogy a Processing hívja a benne készült szoftvereket – két fő részre bontható, a `setup()` és a `draw()` funkcióra. A `setup()` a szoftver indításakor egyszer fut le. Ebben a funkcióban állítható be minden olyan érték, amely a későbbiekben nem fog változni. Itt lehet a kamera beállításait elvégezni. Ehhez kilistáztattam az összes lehetséges kamera beállítást, majd utána kiválasztottam a megfelelőt, ami jelen esetben a 103-as, ezzel egy 1920×1080 pixel méretű kép kérhető le, másodpercenként 30-szor. Lehetőség lett volna nagyobb felbontás használatára, viszont a nagyon alacsony frissítési frekvenciája miatt inkább a kisebb felbontást választottam.

A `draw()` funkció, miután a `setup()` egyszer lefutott, folyamatosan végrehajtásra kerül. Itt van lehetőség, ahogy a neve is utal rá, az ablak tartalmán változtatni. Először a hátterét feketére állítja, majd az alaptérképet jeleníti meg, ami a digitális modellről készült felülnézeti kép. Ehhez a vetített képhez igazítottam a makettet.

A kamera képét is ezen a részen belül dolgozza fel a program.

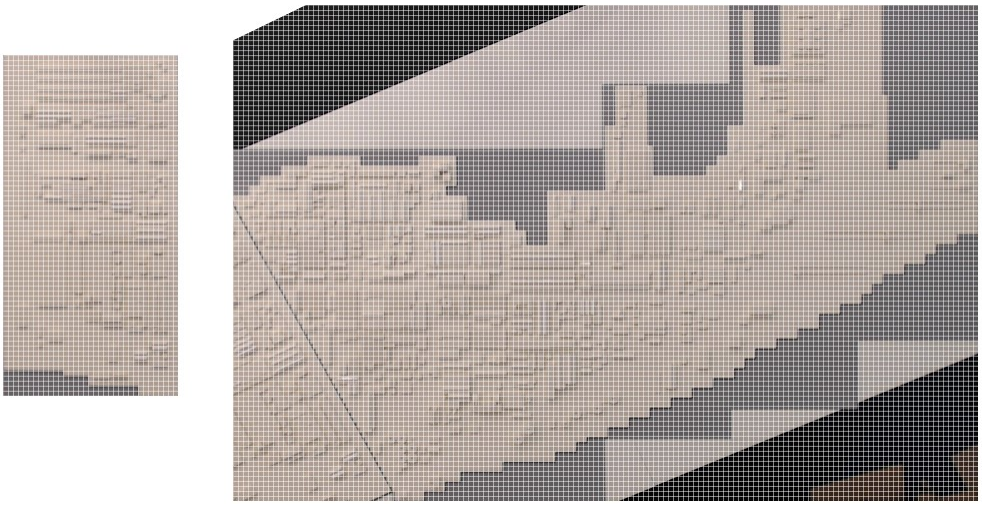
A kamera működéséhez és a vetítéstechnikához egyaránt hozzá tartozik, hogy mivel a fentről nézi a makettet, és a vetített kép is a makettet használja „vászonként”, ezért időnként, megszakítva a vetítést, fehér fénnyel világítja meg a projektor az asztal egészét. Amennyiben nem tenné, a vetített színek meghamisíthatnák a színérzékeléskor az értékeket. Kb. 5 másodpercre szakad meg a vetítés, ezalatt van ideje alkalmazkodni a kamerának a megváltozott fényviszonyokhoz: beállítani a fehéregyensúlyt és a kontrasztot. Mielőtt visszaváltana a vetítésre, a kamera egy fotót készít a modellről, és ezzel dolgozik a szoftver a továbbiakban. Ez, amit „szkennelésnek” hívtunk a fejlesztés során, kb, 15 másodpercenként fut le.

Ahhoz, hogy a két különböző irányú rasztert tudja kezelni, korrigálni kell a torzulásokat, amelyek abból adódnak, hogy a kamera nem pontosan az asztal közepe felett helyezkedik el. Ezzel együtt lehetőség van a raszterek elforgatására is. Az alkalmazott módszer azon alapul, hogy meg kell adni a programnak külön az északi rész 4 sarokpontját, és a déli, elforgatott rész 4 sarokpontját, amik alapján képes lesz egy torzulásmentes helyes irányú képet létrehozni. Ehhez írtam külön egy programot, amely képes volt kiszámítani és kiírni a sarokpontokat.



Kalibrációhoz szükséges sarokpontok kamera által látott helyének megállapítása. A bal oldali képen lehet beállítani, hogy melyik részre nagyított képe jelenjen meg a jobb oldali képen, ahol a kurzor pozícionálásával kapható meg a koordináták értéke.

Ezeket beírva a fő programba, a program két új, korrigált képet készít, egyet az északi részről, és egyet a déli részről. Továbbiakban ezeket használja a számítások során.



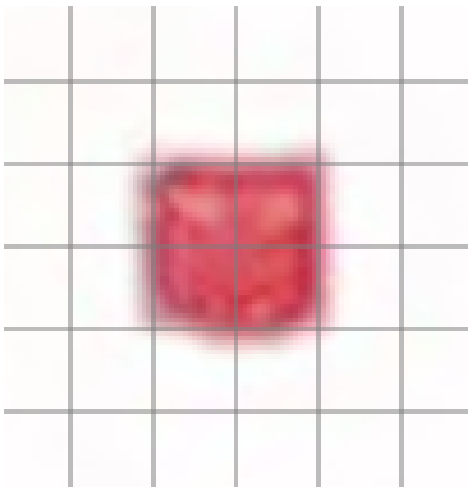
Bal oldalon az északi rész, jobb oldalon a déli rész korrigált képe látható, fehér egyenesek jelzik a rasztert.

Színek érzékelése

A két korrigált kép méretét úgy határoztam meg, hogy egy 2×2-es méretű LEGO elem pontosan 10 pixel szélességű legyen rajta. Mivel a felhasználóknak ennél kisebb színezett elemek nincsenek előkészítve, így ez adja a kép raszterének méretét.

Két egymásba ágyazott ciklus végigpásztázza a képeket, minden, a raszter által kimetszett négyzetben átlagolja a pixelek színét. Kalibrációs hibák elkerülése végett, a négyzeteket

széleinél egy 2 pixel széles sávot figyelmen kívül hagy, így egy 6×6 pixel nagyságú négyzet átlagszínét határozza meg.



A korrigált képen egy piros 4×4-es LEGO elem és szűk környezete.
A 2 LEGO egység (10 pixel) méretű raszter szürke vonalakkal jelölve.

Írtam egy külön programot arra, hogy meg tudjam állapítani, hogy az egyes színezett LEGO elemeknek mi az RGB színkódja. Mind a 4 esetben (kék, zöld, narancssárga, piros) feljegyeztem a színek vörös, zöld és kék tartalmát különböző beállítások és fényviszonyok között. Mind a négy színhez írtam egy függvényt, amely bemeneti értékként egy RGB színkódot kér, tartalmazza az adott szín határértékeit, és visszatérési értékként igaz vagy hamis értéket ad vissza. Pl. a piros esetében, amennyiben a bemeneti szín vörös tartalma 160 és 255, a zöld tartalma 40 és 90, a kék tartalma 60 és 110 közötti érték, igaz értéket ad vissza. Ezt a négy függvényt minden egyes 6×6-os területre lefuttatva, megkapható, hogy hol, milyen színű LEGO elemmel cserélték ki az eredeti elemet.

Adatbázis hozzákapcsolása

Két féle adatbázist képes könnyedén kezelni a Processing, az XML és a JSON formátumokat. Mivel már korábban is használtam a JSON adattáblákat, ezért a fejlesztés során is azt használtam.

Az adattáblát egy kép fájl alapján hoztam létre. A kép tartalmazta a Csepel Művek zajszennyezését, körök középpontjai jelölték a zaj forrását, a kör átmérője a zajszennyezés mértékére utalt. Ahhoz, hogy a több száz zajforrást könnyedén át tudjam ültetni egy adatbázisba, ismét egy külön programot írtam hozzá Processing-ben. A szoftver képként tartalmazta a zajtérképet, valamint, ha lenyomtam a bal egérgombot és elhúztam a mutatót, egy kört rajzolt, úgy, hogy a lenyomás pillanatában vett pozícióban volt a kör középpontja. Az

egérnek ezt a két adatát – a lenyomás pillanatában és felengedéskor vett pozícióját – elmentette egy szöveges fájlban. A zajtérképen lévő összes zajforrás körét ezzel a módszerrel átrajzolva, gyorsan megkaptam a zaj forrásának a helyét és mértékét.

Ezt a szöveges fájlt Excel munkafüzetbe másoltam, amelyből ki tudtam exportálni .csv kiterjesztésű fájlként. Ezt a JSONBuddy nevű programmal némi formázás után .json kiterjesztésű fájlba tudtam konvertálni.

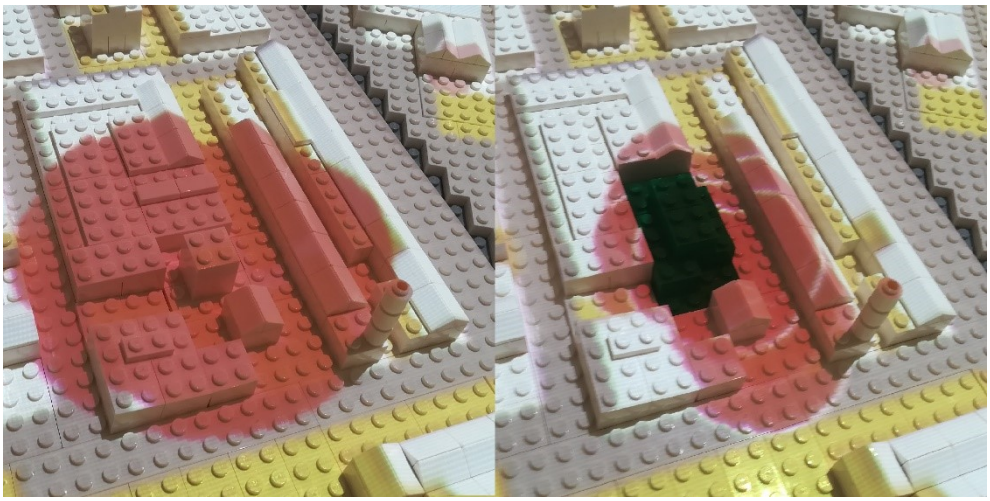
Ezt a fájlt már a Processing képes kezelni. Ezzel a módszerrel többféle adatot is lehet konvertálni.

Algoritmus és vetítés

Már tudja a szoftver, hogy a modellben hol, milyen változtatásokat hajtott végre a felhasználó, és rendelkezésére áll egy adatbázis is. Egy algoritmus ezek alapján különféle eredményeket képes kiszámítani, amely utána már megjeleníthetővé válik.

Először is, a `setup()` funkcióban adható meg, hogy a második képernyőn, tehát a projektoron nyíljon meg az ablak, teljes méretben, keret nélkül. A `draw()` funkcióban állíthatjuk be az alaptérképet mint egy hátteret amire az adatokat ráillesztjük majd.

Az legegyszerűbb algoritmus annyit csinál, hogy megjeleníti az adatbázisból kinyert zaj adatokat, kivéve ott, ahol módosítva van a modell, ott a módosításnak megfelelő információt kell közölnie. Ezzel egészül ki a *Színek érzékelésénél* említett két, egymásba ágyazott ciklus: amennyiben nem érzékel az adott pozícióban színezett elemeket, az adatbázisból kéri le a pozícióhoz tartozó zaj értékét, és jeleníti meg, amennyiben érzékel, az adott színhez előre meghatározott zaj értéket vetíti az adott pontba.



A Csepel Művek egyik legnagyobb zajforrása, és a funkció módosításával kapott új zajérték.

Problémák és megoldások

A fejlesztés során több olyan nehézségbe ütköztünk, melyekre több esetben találtunk megoldást, viszont továbbra is vannak fennálló problémák.

A CityScope technológia egy egyedi gyártású asztalt használ, amely kivitelezése drága és sok időt vesz igénybe. Számunkra ez nem volt járható út, főképp az idő hiánya miatt, ezért újragondolva a rendszert, a teamLab Block Town projektjét alapul véve, a kamerát a modell felett helyeztük el. Ezzel egy olyan megoldást létrehozva, amely használható bármilyen vízszintes, egyenes felületen.

A további problémát a környező fényforrások fel-le kapcsolása, és az ablakon beszűrődő napfény intenzitásának változása jelentette. Lehetőség lett volna a kiállítóterem teljes leárnyékolására, viszont a teljes sötétség az érdeklődők számára valószínűleg kellemetlen lett volna. Ennek a tényezőnek a kiküszöbölése érdekében, *A szoftver felépítése* fejezetben leírt módon, megkerestem azokat a határértékeket, melyeken belül mozognak az érzékelni kívánt színek. Ezeket a minimum és maximum értékeket beírva a szoftverbe, viszonylag stabilan működött a rendszer.

Ehhez hasonlóan a projektor által, a szkenneléskor vetített fehér fény egyenlőtlen eloszlása is meghamisította az eredményeket, amihez még hozzátársult a LEGO elemek fényes felülete, ami remekül tükrözte a projektor fényét.

A projektor által vetített fehér fény meghamisítja az egyes elemek érékelhető színeit. Látható, hogy a balról jobbra haladva, a kék LEGO elem RGB értékei egyre csökkennek, ezzel egyre sötétebb színeket jelezve.

Szoftveresen volt lehetőség ennek a problémának a kezelésére is. A jövőben jobb megoldást jelenthetne, ha a projektor helyett olyan szoftveresen vezérelhető világítás lépne működésbe a szkennelés idejére, ami szórt, egységes mennyiségű fényt juttatna a modellre. Ezzel talán a LEGO elemek visszacsillanása is megakadályozható lenne.

Ami a Csepel Művek esetében nem jelentett gondot, viszont egy nagyobb építménymagassággal rendelkező terület esetén mindenképp problémát jelentene, az a modellépületek képének perspektív torzulása. A makett szélén lévő elemek, amennyiben elérnek egy bizonyos magasságot, a kamera képén úgy tűnhet, mintha a raszter másik koordinátáiba esnének. Erre a problémára megoldást jelenthet a kamera távolabb helyezése, amivel csökkenthető a perspektíva hatása, vagy kisebb terület esetén a Kinect technológia használata, amellyel a LEGO elemek térbeli elhelyezkedése is vizsgálható lenne.

Csökkenthető lehet a szkennelés ideje, a kamera beállításainak rögzítésével. Erre sajnos nem tudtam megoldást találni, annak ellenére, hogy a fókusztávolság, fehéregyensúly és kontraszt beállításai beállíthatók egyedi értékekre, ez csak egy külső programon keresztül végezhető el, ami nem tudja eltárolni a beállításokat. Valószínűleg a kamera zárt API-jal rendelkezik, ami annyit takar, hogy a gyártó nem engedi, csak a saját szoftverének a beállítások módosítását, így nincs lehetőség a Processing környezetén keresztül változtatásokat elvégezni.

Konklúzió

A tervezés, a fejlesztés és az építés kb. másfél hónapot vett igénybe, amit egy kéthetes kiállítás zárt. Ez idő alatt folyamatosan megtekinthető volt bárki számára az interaktív modell, a KÉK kiállítótermében. Mivel a IV. Nemzetközi Építészeti Makettfesztiválhoz kapcsolódóan készítettük, az eseményt is ehhez kapcsoltuk, de mivel a fesztiválra csak 2019 tavaszán kerül sor, ezért mint egy preview-t jelöltük meg. A kiállítás 2018. október 3. és október 19. között volt látható.

Az kiállítást megtekintették a Csepel Művek jelenlegi cégtulajdonosai, vállalkozói, volt dolgozói, városépítészek és hallgatók is. Mindenkitől nagyon pozitív visszajelzéseket kaptunk.

Október 13-án workshop keretében mutattuk be részleteiben a modell működését, majd kezdtünk el ötleteket összeszedni arra vonatkozóan, hogy milyen fejlesztésekkel lehetne több funkcióval felruházni az eszközt. Ennek során elkészítettük egy olyan verzióját, amely képes volt a jelenleg eladó ingatlanok értékét megváltoztatni, amennyiben a környezetébe helyez a felhasználó egy nagyobb zajforrást, vagy csökkent a meglévő zajforrások erősségén.

A résztvevők a nyitóesemény alkalmával először tekinthették meg a modellt működés közben. – forrás: facebook.com/KortarsEpiteszetiKozpont

Fejlesztési lehetőségek

A rendszer stabil működésének megoldása elsődleges, ide tartozik a külső fényforrások hatásának minimalizálása és a kamera érzékelésének javítása.

A jövőben, a meglévő adatok adatbázisban való rendszerezése tudná a modellt rengeteg funkcióval felruházni. Képet lehetne kapni például arról, hogy az utak állapotának javulása, milyen új funkciók betelepülését indukálná, és ez milyen további változásokkal járna. Ugyanígy az úthálózat minőségének romlása esetén, megismerhetők lennének a veszélyek olyan iparágak számára, amelyeknek szükségük van az utakra áruszállítás miatt. Az adatok típusától függően különböző kérdésköröket lehetne körüljárni.

Eredmények

Elmondható, hogy a „cetlis” participációs programokkal ellentétben, a dolgozatban bemutatott eszköz segítségével, a résztvevők ki is próbálhatják elképzeléseiket. Amennyiben a visszajelzés alapján úgy látják, hogy a beavatkozás káros volt, ki tudnak próbálni más elképzeléseket is, amíg megtalálják a legoptimálisabbat. Ezzel sokkal látványosabban, érthetőbben és hatékonyabban lehet az érdekeltek közreműködésével új víziókat, jövőképeket kialakítani.

Hivatkozások

Alonso, L., Zhang, Y. R., Grignard, A., Noyman, A., Sakai, Y., ElKatsha, M., . . . Larson, K. (2018). *Cityscope: a data-driven interactive simulation tool for urban design.*

Grignard, A., Macia, N., Pastor, L. A., Noyman, A., Zhang, Y., & Larson, K. (2018). *Cityscope andorra: a multi-level interactive and tangible agent-based visualization.*

Hanzl, M. (2007). *Information technology as a tool for public participation in urban planning: a review of experiments and potentials,.*

Noyman, A., Holtz, T., Kröger, J., Noennig, J. R., & Larson, K. (2017). *Finding Places: HCI Platform for Public Participation in Refugees' Accommodation Process.*

Pickett, S. T., Burch, W. R., Dalton, S. E., Foresman, T. W., Groove, J. M., & Rowntree, R. (1997). *A conceptual framework for the study of human ecosystems in urban areas.*

Reas, C., & Fry, B. (2007). *Processing: a programming handbook for visual designers and artists.*

Sain, M. (2014). *Mini segédlet a közösségi tervezéshez.*

Sain, M., & Rab, J. (2018). *Részvételi tervezés a településfejlesztésben és településrendezésben.*

von Heland, F., Westerberg, P., & Nyberg, M. (2015). *Using Minecraft as a citizen participation tool in urban design and decision making.*

Melléklet - forráskód

```
JSONObject north_noise;
JSONObject south_noise;

PImage terkep;
PImage src;
PImage dewarped_1;
PImage dewarped_2;

import gab.opencv.*;
import org.opencv.imgproc.Imgproc;
import org.opencv.core.MatOfPoint2f;
import org.opencv.core.Point;
import org.opencv.core.Size;
import org.opencv.core.Mat;
import org.opencv.core.CvType;
import processing.video.*;

Capture cam;
OpenCV opencv;
Contour contour;

float _1_topright_x = 472;
float _1_topright_y = 141;
float _1_topleft_x = 209;
float _1_topleft_y = 142;
float _1_bottomleft_x = 236;
float _1_bottomleft_y = 686;
float _1_bottomright_x = 493;
float _1_bottomright_y = 683;

float _2_topright_x = 1679;
float _2_topright_y = 389;
float _2_topleft_x = 658;
float _2_topleft_y = -148;
float _2_bottomleft_x = 296;
float _2_bottomleft_y = 623;
float _2_bottomright_x = 1303;
float _2_bottomright_y = 1095;

int _1_col = 28;
int _1_row = 64;
int _1_detail_x = 10;
int _1_detail_y = 10;

int _2_col = 136;
int _2_row = 98;
int _2_detail_x = 10;
int _2_detail_y = 10;

int calibrated = 0;
int flag = 0;

float time = 250;

import gifAnimation.*;

PImage[] animation;
Gif loopingGif;

void setup() {
  north_noise = loadJSONObject("north_noise.json");
  south_noise = loadJSONObject("south_noise.json");
  terkep = loadImage("terkep.jpg");
  fullScreen(2);
```

```
  loopingGif = new Gif(this, "scanning.gif");
  loopingGif.loop();

  String[] cameras = Capture.list();

  cam = new Capture(this, cameras[103]);

  for (int i=0; i<cameras.length; i++) {
    println(i, " - ", cameras[i]);
  }

  cam.start();
  src = cam;

  opencv = new OpenCV(this, 1920, 1080);
}

void draw() {
  background(0);
  noStroke();

  image(terkep, 0, 0, 1920, 1080);

  opencv.blur(1);
  opencv.threshold(120);

  if (cam.available() == true) {
    cam.read();
  }

  dewarped_1 = createImage(_1_col*_1_detail_x, _1_row*_1_detail_y, ARGB);
  opencv.toPImage(warpPerspective_1(), dewarped_1);

  dewarped_2 = createImage(_2_col*_2_detail_x, _2_row*_2_detail_y, ARGB);
  opencv.toPImage(warpPerspective_2(), dewarped_2);

  //CHECK FOR COLORED BRICKS AND DRAW CIRCLES

  translate(1920, 1080);
  rotate(PI);

  pushMatrix();
  translate(5, 0);

  for (int i=-2; i<_1_col-2; i++) {
    for (int j=-2; j<_1_row-2; j++) {
      PImage newImg = dewarped_1.get(i*_1_detail_x, j*_1_detail_y, _1_detail_x,
_1_detail_y);

      stroke(255);
      strokeWeight(1);

      flag = 0;

      if (isRed(newImg) == true) {
        fill(255, 0, 0, 200);
        ellipse(map(i, 0, _1_col, 0, 370)+5, map(j, 0, _1_row, 0, 800)+5, 100,
100);
        flag = 1;
      }

      if (isOrange(newImg) == true) {
        fill(255, 0, 0, 165);
        ellipse(map(i, 0, _1_col, 0, 370)+5, map(j, 0, _1_row, 0, 800)+5, 70, 70);
        flag = 1;
      }

      if (isGreen(newImg) == true) {
```

```
        fill(255, 0, 0, 130);
        ellipse(map(i, 0, _1_col, 0, 370)+5, map(j, 0, _1_row, 0, 800)+5, 50, 50);
        flag = 1;
      }

      if (isBlue(newImg) == true) {
        fill(255, 0, 0, 95);
        ellipse(map(i, 0, _1_col, 0, 370)+5, map(j, 0, _1_row, 0, 800)+5, 40, 40);
        flag = 1;
      }

      if (north_noise.hasKey(str(i)) &&
north_noise.getJSONObject(str(i)).hasKey(str(j)) && flag == 0) {
        int a =
north_noise.getJSONObject(str(i)).getJSONObject(str(j)).getInt("value");
        fill(255, 0, 0, map(a, 0, 80, 50, 180));
        noStroke();
        ellipse(map(i, 0, _1_col, -50, 370)+5, map(j, 0, _1_row, -10, 780)+5,
a*1.5, a*1.5);
      }
    }
  }
  popMatrix();

  pushMatrix();
  translate(630, -380);
  rotate(26.5*PI/180);

  for (int i=-2; i<_2_col-2; i++) {
    for (int j=-2; j<_2_row-2; j++) {
      PImage newImg = dewarped_2.get(i*_2_detail_x, j*_2_detail_y, _2_detail_x,
_2_detail_y);
      flag = 0;

      stroke(255);
      strokeWeight(1);

      if (isRed(newImg) == true) {
        fill(255, 0, 0, 200);
        ellipse(map(i, 0, _2_col, 0, 1640)+5, map(j, 0, _2_row, 0, 1190)+5, 100,
100);
        flag = 1;
      }

      if (isOrange(newImg) == true) {
        fill(255, 0, 0, 165);
        ellipse(map(i, 0, _2_col, 0, 1640)+5, map(j, 0, _2_row, 0, 1190)+5, 70,
70);
        flag = 1;
      }

      if (isGreen(newImg) == true) {
        fill(255, 0, 0, 130);
        ellipse(map(i, 0, _2_col, 0, 1640)+5, map(j, 0, _2_row, 0, 1190)+5, 50,
50);
        flag = 1;
      }

      if (isBlue(newImg) == true) {
        fill(255, 0, 0, 95);
        ellipse(map(i, 0, _2_col, 0, 1640)+5, map(j, 0, _2_row, 0, 1190)+5, 40,
40);
        flag = 1;
      }

      if (south_noise.hasKey(str(i)) &&
south_noise.getJSONObject(str(i)).hasKey(str(j)) && flag == 0) {
```

```
        int a =
south_noise.getJSONObject(str(i)).getJSONObject(str(j)).getInt("value");
        fill(255, 0, 0, map(a, 0, 80, 50, 180));
        ellipse(map(i, 0, _2_col, 0, 1640)+5, map(j, 0, _2_row, 0, 1190)+5, a*1.5,
a*1.5);
      }
    }
  }
  popMatrix();

  //INSTRUCTIONS

  translate(1920, 1080);
  rotate(PI);

  textSize(20);
  fill(255);
  text("By replacing the bricks, the function of that building changes.", 350,
980);
  text("The color means a different function, so it has different noise output.",
350, 1010);
  textSize(15);
  stroke(255);
  strokeWeight(1);
  fill(255, 0, 0);
  rect(350, 1030, 30, 30);
  fill(255);
  text("Heavy industry", 390, 1050);
  fill(255, 165, 0);
  rect(530, 1030, 30, 30);
  fill(255);
  text("Light industry", 570, 1050);
  fill(0, 0, 255);
  rect(720, 1030, 30, 30);
  fill(255);
  text("Service and commerce", 760, 1050);
  fill(0, 255, 0);
  rect(960, 1030, 30, 30);
  fill(255);
  text("Storage and logistics", 1000, 1050);


  //CAPTURE IMAGE AFTER 250 CYCLES FOR 100 CYCLES

  if (time >= 250) {
    noStroke();
    fill(255);
    rect(0, 0, 1920, 1080);

    textSize(20);
    fill(0);

    image(loopingGif, 350, 880, 200, 200);

    if (time>=350) {
      time = 0;

      if (cam.width > 0 && cam.height > 0) {
        src = cam;
        opencv.loadImage(src);
      }
    }
  }

  time++;
}

Mat getPerspectiveTransformation_1() {
```

```
  Point[] canonicalPoints = new Point[4];
  canonicalPoints[0] = new Point(_1_col*_1_detail_x, 0);
  canonicalPoints[1] = new Point(0, 0);
  canonicalPoints[2] = new Point(0, _1_row*_1_detail_y);
  canonicalPoints[3] = new Point(_1_col*_1_detail_x, _1_row*_1_detail_y);

  MatOfPoint2f canonicalMarker = new MatOfPoint2f();
  canonicalMarker.fromArray(canonicalPoints);

  Point[] points = new Point[4];
  points[0] = new Point(_1_topright_x, _1_topright_y);
  points[1] = new Point(_1_topleft_x, _1_topleft_y);
  points[2] = new Point(_1_bottomleft_x, _1_bottomleft_y);
  points[3] = new Point(_1_bottomright_x, _1_bottomright_y);

  MatOfPoint2f marker = new MatOfPoint2f(points);
  return Imgproc.getPerspectiveTransform(marker, canonicalMarker);
}

Mat warpPerspective_1() {
  Mat transform = getPerspectiveTransformation_1();
  Mat unWarpedMarker = new Mat(_1_col*_1_detail_x, _1_row*_1_detail_y,
CvType.CV_8UC1);
  Imgproc.warpPerspective(opencv.getColor(), unWarpedMarker, transform, new
Size(_1_col*_1_detail_x, _1_row*_1_detail_y));
  return unWarpedMarker;
}

Mat getPerspectiveTransformation_2() {
  Point[] canonicalPoints = new Point[4];
  canonicalPoints[0] = new Point(_2_col*_2_detail_x, 0);
  canonicalPoints[1] = new Point(0, 0);
  canonicalPoints[2] = new Point(0, _2_row*_1_detail_y);
  canonicalPoints[3] = new Point(_2_col*_2_detail_x, _2_row*_1_detail_y);

  MatOfPoint2f canonicalMarker = new MatOfPoint2f();
  canonicalMarker.fromArray(canonicalPoints);

  Point[] points = new Point[4];
  points[0] = new Point(_2_topright_x, _2_topright_y);
  points[1] = new Point(_2_topleft_x, _2_topleft_y);
  points[2] = new Point(_2_bottomleft_x, _2_bottomleft_y);
  points[3] = new Point(_2_bottomright_x, _2_bottomright_y);

  MatOfPoint2f marker = new MatOfPoint2f(points);
  return Imgproc.getPerspectiveTransform(marker, canonicalMarker);
}

Mat warpPerspective_2() {
  Mat transform = getPerspectiveTransformation_2();
  Mat unWarpedMarker = new Mat(_2_col*_1_detail_x, _2_row*_1_detail_y,
CvType.CV_8UC1);
  Imgproc.warpPerspective(opencv.getColor(), unWarpedMarker, transform, new
Size(_2_col*_1_detail_x, _2_row*_1_detail_y));
  return unWarpedMarker;
}

boolean isRed(PImage img) {
  img.loadPixels();
  int r = 0, g = 0, b = 0;
  for (int i=0; i<img.pixels.length; i++) {
    color c = img.pixels[i];
    r += c>>16&0xFF;
    g += c>>8&0xFF;
    b += c&0xFF;
  }
  r /= img.pixels.length;
  g /= img.pixels.length;
```

```
  b /= img.pixels.length;

  if (r>160 && g<90 && g>40 && b<110 && b>60) {
    return true;
  } else {
    return false;
  }
}

boolean isOrange(PImage img) {
  img.loadPixels();
  int r = 0, g = 0, b = 0;
  for (int i=0; i<img.pixels.length; i++) {
    color c = img.pixels[i];
    r += c>>16&0xFF;
    g += c>>8&0xFF;
    b += c&0xFF;
  }
  r /= img.pixels.length;
  g /= img.pixels.length;
  b /= img.pixels.length;

  if (r>210 && g>120 && g<190 && b<120) {
    return true;
  } else {
    return false;
  }
}

boolean isGreen(PImage img) {
  img.loadPixels();
  int r = 0, g = 0, b = 0;
  for (int i=0; i<img.pixels.length; i++) {
    color c = img.pixels[i];
    r += c>>16&0xFF;
    g += c>>8&0xFF;
    b += c&0xFF;
  }
  r /= img.pixels.length;
  g /= img.pixels.length;
  b /= img.pixels.length;

  if (r>20 && r<90 && g>90 && g<160 && b>80 && b<140) {
    return true;
  } else {
    return false;
  }
}

boolean isBlue(PImage img) {
  img.loadPixels();
  int r = 0, g = 0, b = 0;
  for (int i=0; i<img.pixels.length; i++) {
    color c = img.pixels[i];
    r += c>>16&0xFF;
    g += c>>8&0xFF;
    b += c&0xFF;
  }
  r /= img.pixels.length;
  g /= img.pixels.length;
  b /= img.pixels.length;

  if (r<90 && g<200 && g>60 && b>140) {
    return true;
  } else {
    return false;
  }
}
```