Computational Portfolio

Ziyuan Gao

Data Network in Python

Grab geodata directly from the web : display dynamics web maps inside Blender 3d view, requests for OpenStreetMap data (buildings, roads ...), get true elevation data from the NASA SRTM mission.

STEP1: Define a class of ladefensebuildings with its various attributes including the building's name, function, etc. In Python every object is an instance of a class. The class defines the characteristics an object.

```
class ladefensebuilding:
    def __init__(self,name,function,height,delivery,zone,node,longitude,latitude):
        self.building = building
        self.function = function
        self.height = height
        self.delivery = delivery
        self.zone = zone
        self.node = node
        self.longitude = longitude
        self.latitude = latitude
```

STEP2: Import and read csv files

	ladefense	Building	longitude	latitude	Location	Surface a	Offices(m	Shops(m²	height	function	Delivery	zone	node
2		Seasons (	48.88943679	2.251162198	4, Place d	8300			69	Hotel	1993	square de	esplande nord
3		Melia Par	48.8885475	2.250572641	3 Boulevard	de Neuilly, 92400 Courbev			38	Hotel	1976	Place des	esplande nord
4		Manhatta	48.88978887	2.248526063	40-42 terrasse	de l'Iris 92400 Courbevc			38	Hotel	1976	Place de l	esplande nord
5		Sofitel De	48.888289	2.247445	33,Voie des	Sculpteurs - 92800 Puteau:			34	Hotel	1985	Voie des	esplande sud
6		Pullman	48.894949	2.238641	11,Avenue	de l'Arche - 92400 Courbev			35	Hotel	1990	Avenue d	arche nord
7		Defense F	48.887603	2.244125	23-27,	F	31100		54	Office	2004	Rue Delar	esplande sud
8		Neptune	48.88949956	2.254110907	20, place		50,500		117	Office	1975	place de	esplande nord
9		First	48.88876393	2.251601172	1 2, Place		86000		231	Office	2011	square de	esplande nord
10		Initiale	48.88704	2.25153	1,Terrasse		34300		105	Office	2003	Terrasse l	esplande sud
11		Miroirs (L	48.89040072	2.250399427	18, Avenu		57500		69	Office	1981	Avenue A	esplande nord
12		CB21	48.88887468	2.249909194	Esplanad	78 000			188	Office	2021	Place de l	esplande nord
13		Allianz Or	48.88761	2.24928	1,cours M		38000		85	Office	2015	cours Mic	esplande sud
14		Saint Gob	48.88948957	2.249256331	La Défens	48 948			165	Office	2020	Place de l	esplande nord
15		Balzac	48.89191315	2.248813191	10, Place		22700		35	Office	1989	Place des	esplande nord
16		La Fayett	48.8930583	2.248650977	2-3, Place		15500		35	Office	1989	Place des	esplande nord
17		Lavoisier	48.89310023	2.248570648	4, Place d		22510		35	Office	1989	Place des	esplande nord
18		Manhatta	48.88997939	2.248511716	5-6, Place		80100		110	Office	1975	Place de l	esplande nord
19		Allianz Ac	48.88805	2.24839	13,esplan		30700		34	Office	1985	esplanad	esplande sud
20		Newton	48.89231799	2.247902821	9, Place d		22700		35	Office	1989	Place des	esplande nord
21		D2	48.89051424	2.247879812	17 bis Pla		54530	49000	175	Office	2015	Place des	esplande nord
22		Monge	48.89147549	2.247356238	22, Place		9500		34	Office	1996	Place des	esplande nord
23		Aurore	48.89014587	2.247345612	18, Place	38 855		32779	1 595	100	Office	1970	Place des	esplande nord
24		CB16-CG	48.890551	2.247167971	17, Place		28000			110	Office	2006	Place des	esplande nord
25		Egho	48.09229371	2.246763496	2, Avenue		77,000	79,000		131	Office	2012	Avenue A	esplande nord
26		Euronext	48.89064687	2.246738002	7, Terras		10000			35	Office	2009	Place des	esplande nord

```
with open("ladefensebuildingfunction.csv", encoding='mac_roman') as csvfile:
    reader = csv.reader(csvfile)
    rows = 0
    for r in reader:
        rows = rows + 1
```

```
ladefensebuildings=[]
with open("ladefensebuildingfunction.csv", encoding='mac_roman') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        building = row['Building']
        function = row['function']
        height = int(row['height'])
        delivery = int(row['Delivery'])
        zone = row['zone']
        node = row['node']
        longitude = float(row['longitude'])
        latitude = float(row['latitude'])
        ladefensebuildings.append(ladefensebuilding(building,function,height,delivery,zone,node,longitude,lat
```

STEP3: Assign latitude and longitude to each node

```
for P in ladefensebuildings:
    ladefensebuildinggraph.add_node(P, pos = (P.longitude, P.latitude))
```

```
position=nx.get_node_attributes(ladefensebuildinggraph,'pos')
```

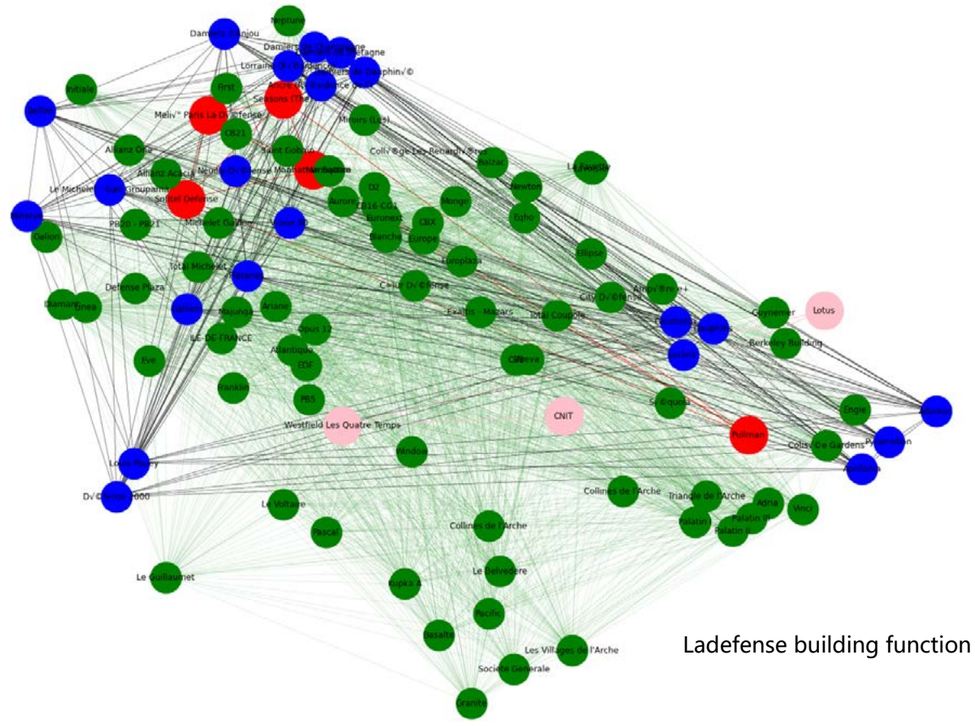

STEP4: Distinguish each node in terms of its function(or other attributes). Add edge to nodes of same function

```
for P1 in ladefensebuildings:
    for P2 in ladefensebuildings:
        if not P1 == P2:
            if P1.function == P2.function=='Hotel':
                    ladefensebuildinghotelgraph.add_edge(P1,P2)
                    ladefensebuildinghotelgraph.add_node(P1, pos = (P1.longitude, P1.latitude))

            if P1.function == P2.function=='Office':
                    ladefensebuildingofficegraph.add_edge(P1,P2)
                    ladefensebuildingofficegraph.add_node(P1, pos = (P1.longitude, P1.latitude))

            if P1.function == P2.function=='Residence':
                    ladefensebuildingresidencegraph.add_edge(P1,P2)
                    ladefensebuildingresidencegraph.add_node(P1, pos = (P1.longitude, P1.latitude))

            if P1.function == P2.function=='Shop':
                    ladefensebuildingshopgraph.add_edge(P1,P2)
                    ladefensebuildingshopgraph.add_node(P1, pos = (P1.longitude, P1.latitude))
```

STEP5: Visualize the graph and label each node

```
pos = nx.spring_layout(ladefensebuildinggraph)
node_labels = {}
for node in ladefensebuildinggraph.nodes:
    node_labels[node] = str(node.building)
```

```
optionoffice={'node_color':'green','node_size':2000, 'width':0.1, 'edge_color':'green'}
optionhotel={'node_color':'red','node_size':4000, 'width':0.8, 'edge_color':'red'}
optionresidence={'node_color':'blue','node_size':3000, 'width':0.5, 'edge_color':'black'}
optionshop={'node_color':'pink','node_size':3000, 'width':1, 'edge_color':'pink'}

plt.figure(1, figsize = (20,20))

nx.draw(ladefensebuildinghotelgraph,position, **optionhotel)
nx.draw(ladefensebuildingofficegraph,position, **optionoffice)
nx.draw(ladefensebuildingresidencegraph,position, **optionresidence)
nx.draw(ladefensebuildingshopgraph,position, **optionshop)
nx.draw_networkx_labels(ladefensebuildinggraph, position, labels=node_labels)
```



Ladefense building function

Scrape image data from flickr with python

The project focuses on building a web scraper for collecting photographs and the key elements behind such as topics, equipment used, camera settings, geographic locations, etc. The collected data will then be analyzed and visualized to give the intuition of which kinds of equipment are applicable for given topics.

```
def download_file(url, local_filename):
    if local_filename is None:
        local_filename = url.split('/')[-1]
    r = requests.get(url, stream=True)
    with open(local_filename, 'wb') as f:
        for chunk in r.iter_content(chunk_size=1024):
            if chunk:
                f.write(chunk)
    return local_filename
```

```
def get_group_id_from_url(url):
    params = {
        'method' : 'flickr.urls.lookupGroup',
        'url': url,
        'format': 'json',
        'api_key': KEY,
        'format': 'json',
        'nojsoncallback': 1
    }
    results = requests.get('https://api.flickr.com/services/rest', params=params).json()
    return results['group']['id']
```

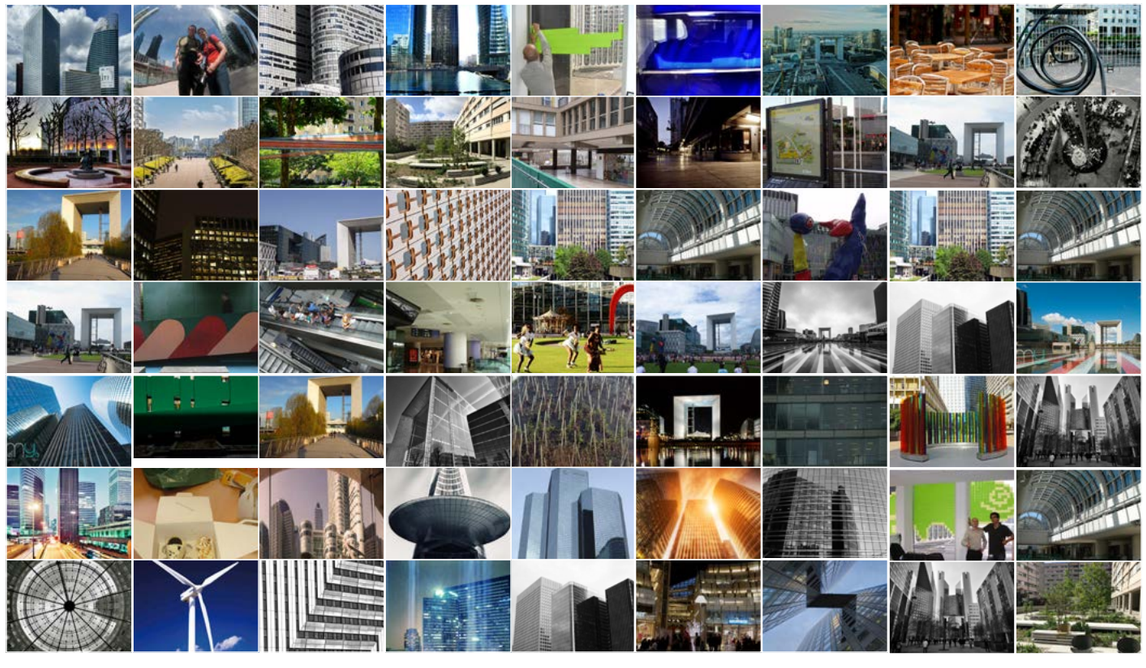
```
def get_photos(qs, qg, page=1, original=False, bbox=None):
    params = {
        'content_type': '7',
        'per_page': '500',
        'media': 'photos',
        'format': 'json',
        'advanced': 1,
        'nojsoncallback': 1,
        'extras': 'media,realname,%s,o_dims,geo,tags,machine_tags,date_taken' % ('url_o' if original else 'u
        'page': page,
        'api_key': KEY
    }

    if qs is not None:
        params['method'] = 'flickr.photos.search',
        params['text'] = qs
    elif qg is not None:
        params['method'] = 'flickr.groups.pools.getPhotos',
        params['group_id'] = qg

    # bbox should be: minimum_longitude, minimum_latitude, maximum_longitude, maximum_latitude
    if bbox is not None and len(bbox) == 4:
        params['bbox'] = ','.join(bbox)

    results = requests.get('https://api.flickr.com/services/rest', params=params).json()
    if "photos" not in results:
        print(results)
        return None
    return results["photos"]
```

```
def search(qs, qg, bbox=None, original=False, max_pages=None,start_page=1):
    # create a folder for the query if it does not exist
    foldername = os.path.join('images', re.sub(r'[\W]', '_', qs if qs is not None else "group_%s"%qg))
    if bbox is not None:
        foldername += '_'.join(bbox)

    if not os.path.exists(foldername):
        os.makedirs(foldername)

    jsonfilename = os.path.join(foldername, 'results' + str(start_page) + '.json')
```



```
    if not os.path.exists(jsonfilename):

        # save results as a json file
        photos = []
        current_page = start_page

        results = get_photos(qs, qg, page=current_page, original=original, bbox=bbox)
        if results is None:
            return

        total_pages = results['pages']
        if max_pages is not None and total_pages > start_page + max_pages:
            total_pages = start_page + max_pages

        photos += results['photo']

        while current_page < total_pages:
            print('downloading metadata, page {} of {}'.format(current_page, total_pages))
            current_page += 1
            photos += get_photos(qs, qg, page=current_page, original=original, bbox=bbox)['photo']
            time.sleep(0.5)

        with open(jsonfilename, 'w') as outfile:
            json.dump(photos, outfile)

    else:
        with open(jsonfilename, 'r') as infile:
            photos = json.load(infile)

    # download images
    print('Downloading images')
    for photo in tqdm(photos):
        try:
            url = photo.get('url_o' if original else 'url_l')
            extension = url.split('.')[-1]
            localname = os.path.join(foldername, '{}.{}'.format(photo['id'], extension))
            if not os.path.exists(localname):
                download_file(url, localname)
        except Exception as e:
            continue
```

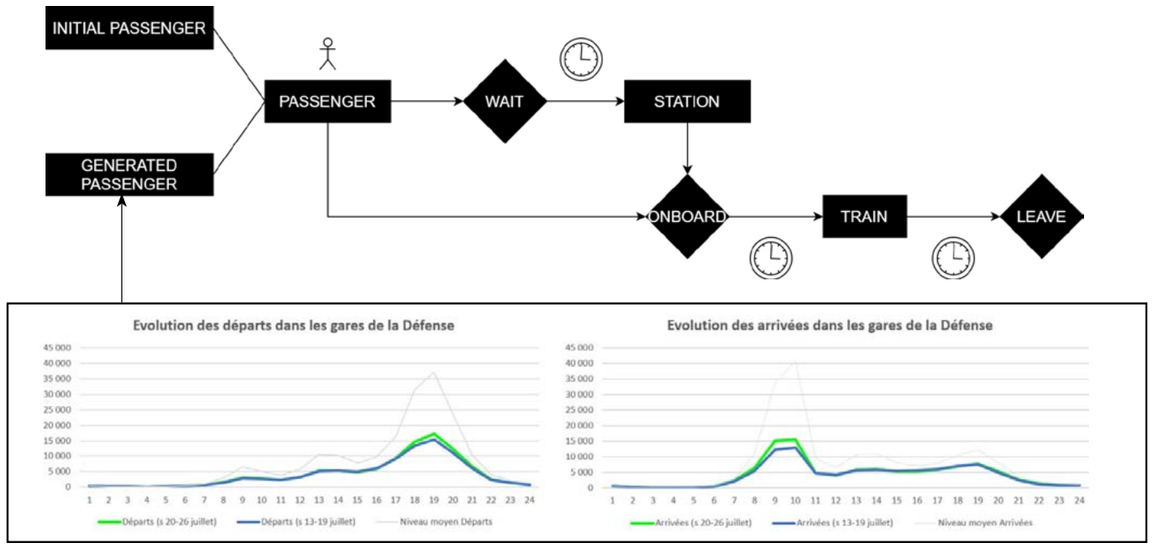
Tagged photo of Ladefense scraping from Flickr



Pedestrian simulation in Python

A simulation is an approximate imitation of the operation of a process or system that represents its operation over time.

Ladefense train station simulation Scenario:
A station system has a limited number of trains and defines a onboard processes.Passenger processes arrive at the station based on the data of the departing and arriving flow. If one train is available, they start the onboard process, the train will stay in the station for 1 minute and leave.If not, they wait until next one.The interval time between each train is 3 minutes.



STEP1: Import csv files. Define a class of ladefense train station with its various attributes including the number of people departing and arrving per hour.

```
NUM_TRAINS = 2
WAITTIME = 3
ONBOARDTIME = 1
T_INTER = 1
SIM_TIME = 1440
```

```
class Flow:
    def __init__(self,time,arrive,depart,flowall):
        self.time = time
        self.arrive = arrive
        self.depart = depart
        self.flowall = flowall
```

```
with open("flow.csv", encoding='mac_roman') as csvfile:
    reader = csv.reader(csvfile)
    rows = 0
    for r in reader:
        rows = rows + 1
```

```
Flows=[]
with open("flow.csv", encoding='mac_roman') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        hour = int(row['time'])
        arrive = row['arrive']
        depart = row['depart']
        flowall = int(row['flowall'])
        Flows.append(Flow(hour,arrive,depart,flowall))
```


STEP2: Define onboard class. A station has a limited number of trains (``NUM_NUM_TRAINS``) to carry passengers in parallel. Passengerss have to request one of the Trains. When they get on one train, they can start the onboard processes and wait 1 minute for it to leave the station (which takes ``onboardtime`` minutes).

```
class Onboard(object):
    def __init__(self, env, num_trains, waittime, onboardtime):
        self.env = env
        self.train = simpy.Resource(env, num_trains)
        self.onboardtime = onboardtime
        self.waittime = waittime
```

The waiting function. It takes a group of``passengers`` processes and tries to wait for it.

```
    def wait(self, passengers):
        yield self.env.timeout(WAITTIME)
```

The passenger function. Each passenger (has a ``name``) arrives at the station (``cw``) and requests a train. It then starts the onboard process, waits for it to finish and leaves to never come back.

```
def passenger(env, name, cw):
    hour=int(env.now/60)
    minute=env.now-hour*60

    print('%s arrives at the station at %s:%s' % (name, hour, minute))
    with open('document7.csv','a') as fd:
        fields=[name,'%s:%s' % (hour, minute)]
        writer = csv.writer(fd)
        writer.writerow(fields)
        fd.close()
    with cw.train.request() as request:
        print('%s enters the train at %s:%s' % (name, hour, minute))
        yield env.process(cw.wait(name))
        print('%s leaves the station at %s:%s' % (name, hour, minute))
```

The setup function. Create a onboard, a number of initial passengers and keep creating passengers approx. every ``t_inter`` minutes.

```
def setup(env, num_trains, waittime, t_inter, onboardtime):
    onboard = Onboard(env, num_trains, waittime, onboardtime)
```

Create 10 initial cars.

```
    for i in range(10):
        env.process(passenger(env, 'Passenger %d' % i, onboard))
```

Create more passengers while the simulation is running.

```
    while True:
        yield env.timeout(random.randint(t_inter - 1 , t_inter + 1))
        i += (Flows[int(env.now/60)].flowall)/60
        env.process(passenger(env, 'Passenger %d' % i, onboard))
        print(int(env.now/60))
```

STEP3: Execution and simulation. Create an environment and start the setup process.

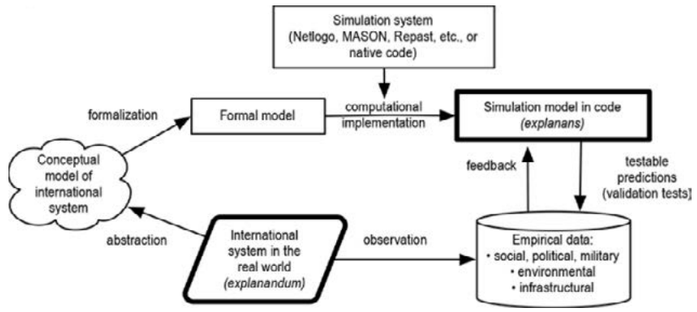
```
env = simpy.Environment()
env.process(setup(env, NUM_TRAINS, WAITTIME, T_INTER, ONBOARDTIME))
env.run(until=SIM_TIME)
```

Passenger simulation of Ladefense train station per minute

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	0:00	Passenger 254	0:58	Passenger 443	2:09	Passenger 585	3:01	Passenger 682	3:59	Passenger 798	4:48	Passenger 931	5:48	Passenger 1248	6:40	Passenger 1765	7:37	Passenger 2718	8:28	Passenger 4135	9:15	Passenger 5882	10:20	Passenger 7218	11:15	Passenger 8185	12:06	Passenger 9153
2	0:00	Passenger 259	0:58	Passenger 445	2:09	Passenger 587	3:03	Passenger 684	4:00	Passenger 800	4:50	Passenger 933	5:48	Passenger 1255	6:40	Passenger 1775	7:37	Passenger 2740	8:28	Passenger 4166	9:17	Passenger 5908	10:22	Passenger 7235	11:16	Passenger 8202	12:07	Passenger 9172
3	0:00	Passenger 262	1:00	Passenger 448	2:09	Passenger 588	3:05	Passenger 686	4:01	Passenger 802	4:50	Passenger 935	5:48	Passenger 1262	6:42	Passenger 1785	7:37	Passenger 2761	8:28	Passenger 4198	9:17	Passenger 5935	10:24	Passenger 7252	11:17	Passenger 8218	12:08	Passenger 9190
4	0:00	Passenger 265	1:02	Passenger 450	2:09	Passenger 590	3:07	Passenger 688	4:02	Passenger 804	4:52	Passenger 938	5:48	Passenger 1268	6:42	Passenger 1795	7:39	Passenger 2783	8:28	Passenger 4230	9:19	Passenger 5962	10:24	Passenger 7268	11:18	Passenger 8235	12:08	Passenger 9208
5	0:00	Passenger 268	1:03	Passenger 453	2:10	Passenger 592	3:08	Passenger 690	4:02	Passenger 806	4:54	Passenger 940	5:48	Passenger 1275	6:44	Passenger 1805	7:41	Passenger 2805	8:28	Passenger 4261	9:20	Passenger 5988	10:26	Passenger 7285	11:20	Passenger 8252	12:10	Passenger 9227
6	0:00	Passenger 272	1:05	Passenger 455	2:11	Passenger 593	3:08	Passenger 692	4:04	Passenger 808	4:55	Passenger 942	5:49	Passenger 1282	6:46	Passenger 1815	7:43	Passenger 2826	8:28	Passenger 4293	9:20	Passenger 6015	10:26	Passenger 7302	11:22	Passenger 8268	12:10	Passenger 9245
7	0:00	Passenger 275	1:05	Passenger 458	2:13	Passenger 595	3:10	Passenger 694	4:04	Passenger 810	4:57	Passenger 945	5:49	Passenger 1288	6:47	Passenger 1825	7:43	Passenger 2848	8:28	Passenger 4325	9:21	Passenger 6042	10:27	Passenger 7318	11:23	Passenger 8285	12:11	Passenger 9263
8	0:00	Passenger 278	1:05	Passenger 460	2:13	Passenger 597	3:10	Passenger 696	4:04	Passenger 812	4:59	Passenger 947	5:49	Passenger 1295	6:49	Passenger 1835	7:43	Passenger 2870	8:28	Passenger 4356	9:23	Passenger 6068	10:28	Passenger 7335	11:24	Passenger 8302	12:12	Passenger 9282
9	0:00	Passenger 282	1:06	Passenger 463	2:13	Passenger 598	3:11	Passenger 698	4:06	Passenger 814	5:01	Passenger 949	5:50	Passenger 1302	6:50	Passenger 1845	7:45	Passenger 2891	8:29	Passenger 4388	9:25	Passenger 6095	10:30	Passenger 7352	11:24	Passenger 8318	12:12	Passenger 9300
10	0:00	Passenger 285	1:06	Passenger 465	2:14	Passenger 600	3:12	Passenger 700	4:06	Passenger 816	5:03	Passenger 952	5:52	Passenger 1308	6:52	Passenger 1855	7:46	Passenger 2913	8:31	Passenger 4420	9:27	Passenger 6122	10:31	Passenger 7368	11:25	Passenger 8335	12:12	Passenger 9318
11	0:00	Passenger 288	1:08	Passenger 468	2:16	Passenger 602	3:12	Passenger 702	4:07	Passenger 819	5:04	Passenger 954	5:53	Passenger 1315	6:52	Passenger 1865	7:48	Passenger 2935	8:33	Passenger 4451	9:27	Passenger 6148	10:32	Passenger 7385	11:25	Passenger 8352	12:12	Passenger 9337
12	0:00	Passenger 292	1:10	Passenger 470	2:16	Passenger 603	3:13	Passenger 704	4:07	Passenger 821	5:06	Passenger 956	5:54	Passenger 1322	6:52	Passenger 1875	7:48	Passenger 2956	8:33	Passenger 4483	9:29	Passenger 6175	10:33	Passenger 7402	11:25	Passenger 8368	12:13	Passenger 9355
13	0:02	Passenger 295	1:10	Passenger 473	2:17	Passenger 605	3:13	Passenger 706	4:09	Passenger 823	5:07	Passenger 959	5:54	Passenger 1328	6:52	Passenger 1885	7:48	Passenger 2978	8:33	Passenger 4515	9:29	Passenger 6202	10:33	Passenger 7418	11:25	Passenger 8385	12:15	Passenger 9373
14	0:04	Passenger 298	1:12	Passenger 475	2:17	Passenger 607	3:15	Passenger 708	4:09	Passenger 826	5:09	Passenger 961	5:55	Passenger 1335	6:52	Passenger 1895	7:50	Passenger 3000	8:35	Passenger 4546	9:30	Passenger 6228	10:35	Passenger 7435	11:25	Passenger 8402	12:16	Passenger 9392
15	0:05	Passenger 302	1:13	Passenger 478	2:18	Passenger 608	3:16	Passenger 710	4:09	Passenger 828	5:11	Passenger 963	5:55	Passenger 1342	6:54	Passenger 1905	7:50	Passenger 3021	8:36	Passenger 4578	9:32	Passenger 6255	10:37	Passenger 7452	11:26	Passenger 8418	12:16	Passenger 9410
16	0:07	Passenger 305	1:13	Passenger 480	2:19	Passenger 610	3:17	Passenger 712	4:10	Passenger 830	5:13	Passenger 966	5:57	Passenger 1348	6:55	Passenger 1915	7:51	Passenger 3043	8:38	Passenger 4610	9:34	Passenger 6282	10:38	Passenger 7468	11:26	Passenger 8435	12:18	Passenger 9428
17	0:08	Passenger 308	1:15	Passenger 483	2:21	Passenger 612	3:17	Passenger 714	4:10	Passenger 833	5:13	Passenger 968	5:58	Passenger 1355	6:57	Passenger 1925	7:51	Passenger 3065	8:40	Passenger 4641	9:36	Passenger 6308	10:38	Passenger 7485	11:27	Passenger 8452	12:19	Passenger 9447
18	0:09	Passenger 312	1:16	Passenger 485	2:23	Passenger 613	3:17	Passenger 716	4:12	Passenger 835	5:13	Passenger 975	6:00	Passenger 1362	6:58	Passenger 1935	7:52	Passenger 3086	8:40	Passenger 4673	9:38	Passenger 6335	10:40	Passenger 7502	11:29	Passenger 8468	12:19	Passenger 9465
19	0:09	Passenger 315	1:16	Passenger 488	2:25	Passenger 615	3:19	Passenger 718	4:12	Passenger 837	5:14	Passenger 981	6:02	Passenger 1368	6:59	Passenger 1945	7:52	Passenger 3108	8:42	Passenger 4705	9:39	Passenger 6362	10:42	Passenger 7518	11:30	Passenger 8485	12:21	Passenger 9483
20	0:09	Passenger 318	1:18	Passenger 490	2:26	Passenger 617	3:19	Passenger 720	4:14	Passenger 840	5:15	Passenger 988	6:03	Passenger 1375	6:59	Passenger 1955	7:52	Passenger 3130	8:44	Passenger 4736	9:40	Passenger 6388	10:42	Passenger 7535	11:32	Passenger 8502	12:23	Passenger 9502
21	0:10	Passenger 322	1:20	Passenger 493	2:28	Passenger 618	3:20	Passenger 722	4:14	Passenger 842	5:16	Passenger 995	6:04	Passenger 1385	7:00	Passenger 1965	7:54	Passenger 3151	8:45	Passenger 4768	9:41	Passenger 6415	10:43	Passenger 7552	11:33	Passenger 8518	12:24	Passenger 9520
22	0:12	Passenger 325	1:22	Passenger 495	2:30	Passenger 620	3:21	Passenger 724	4:15	Passenger 844	5:16	Passenger 1001	6:05	Passenger 1395	7:00	Passenger 1975	7:54	Passenger 3173	8:45	Passenger 4800	9:41	Passenger 6442	10:43	Passenger 7568	11:34	Passenger 8535	12:24	Passenger 9538
23	0:13	Passenger 328	1:23	Passenger 498	2:31	Passenger 622	3:22	Passenger 726	4:17	Passenger 847	5:16	Passenger 1008	6:05	Passenger 1405	7:01	Passenger 1985	7:55	Passenger 3195	8:45	Passenger 4831	9:42	Passenger 6468	10:44	Passenger 7585	11:35	Passenger 8552	12:26	Passenger 9557
24	0:15	Passenger 332	1:23	Passenger 500	2:31	Passenger 623	3:24	Passenger 728	4:17	Passenger 849	5:17	Passenger 1015	6:07	Passenger 1415	7:03	Passenger 1995	7:55	Passenger 3216	8:45	Passenger 4863	9:44	Passenger 6495	10:44	Passenger 7602	11:36	Passenger 8568	12:26	Passenger 9575
25	0:17	Passenger 335	1:24	Passenger 503	2:31	Passenger 625	3:25	Passenger 730	4:18	Passenger 851	5:17	Passenger 1021	6:07	Passenger 1425	7:04	Passenger 2005	7:57	Passenger 3238	8:45	Passenger 4895	9:46	Passenger 6522	10:44	Passenger 7618	11:37	Passenger 8585	12:26	Passenger 9593
26	0:19	Passenger 338	1:25	Passenger 505	2:32	Passenger 627	3:26	Passenger 732	4:19	Passenger 854	5:19	Passenger 1028	6:07	Passenger 1435	7:05	Passenger 2015	7:58	Passenger 3260	8:46	Passenger 4926	9:46	Passenger 6548	10:45	Passenger 7635	11:38	Passenger 8602	12:26	Passenger 9612
27	0:20	Passenger 342	1:25	Passenger 508	2:33	Passenger 628	3:27	Passenger 734	4:19	Passenger 856	5:21	Passenger 1035	6:07	Passenger 1445	7:05	Passenger 2025	7:59	Passenger 3281	8:47	Passenger 4958	9:48	Passenger 6575	10:47	Passenger 7652	11:38	Passenger 8618	12:27	Passenger 9630
28	0:21	Passenger 345	1:27	Passenger 510	2:34	Passenger 630	3:28	Passenger 736	4:19	Passenger 858	5:21	Passenger 1041	6:07	Passenger 1455	7:05	Passenger 2047	8:01	Passenger 3303	8:47	Passenger 4990	9:48	Passenger 6602	10:49	Passenger 7668	11:38	Passenger 8635	12:28	Passenger 9648
29	0:23	Passenger 348	1:28	Passenger 513	2:34	Passenger 632	3:29	Passenger 738	4:19	Passenger 861	5:21	Passenger 1048	6:07	Passenger 1465	7:05	Passenger 2068	8:03	Passenger 3325	8:49	Passenger 5021	9:48	Passenger 6628	10:50	Passenger 7685	11:38	Passenger 8652	12:30	Passenger 9667
30	0:23	Passenger 352	1:28	Passenger 515	2:35	Passenger 633	3:29	Passenger 740	4:19	Passenger 863	5:22	Passenger 1055	6:09	Passenger 1475	7:07	Passenger 2090	8:04	Passenger 3346	8:50	Passenger 5053	9:48	Passenger 6655	10:52	Passenger 7702	11:38	Passenger 8668	12:32	Passenger 9685
31	0:23	Passenger 355	1:30	Passenger 518	2:36	Passenger 635	3:30	Passenger 742	4:19	Passenger 865	5:22	Passenger 1061	6:10	Passenger 1485	7:07	Passenger 2112	8:06	Passenger 3368	8:50	Passenger 5085	9:50	Passenger 6682	10:53	Passenger 7718	11:40	Passenger 8685	12:32	Passenger 9703
32	0:25	Passenger 358	1:31	Passenger 520	2:38	Passenger 637	3:32	Passenger 744	4:21	Passenger 868	5:23	Passenger 1068	6:12	Passenger 1495	7:07	Passenger 2133	8:08	Passenger 3390	8:52	Passenger 5117	9:52	Passenger 6708	10:53	Passenger 7735	11:40	Passenger 8702	12:33	Passenger 9722
33	0:26	Passenger 362	1:33	Passenger 523	2:39	Passenger 638	3:32	Passenger 746	4:23	Passenger 870	5:23	Passenger 1075	6:14	Passenger 1505	7:07	Passenger 2155	8:10	Passenger 3411	8:54	Passenger 5148	9:52	Passenger 6735	10:54	Passenger 7752	11:42	Passenger 8718	12:35	Passenger 9740
34	0:27	Passenger 365	1:34	Passenger 525	2:40	Passenger 640	3:33	Passenger 748	4:24	Passenger 872	5:25	Passenger 1081	6:14	Passenger 1515	7:09	Passenger 2177	8:12	Passenger 3433	8:54	Passenger 5180	9:52	Passenger 6762	10:54	Passenger 7768	11:44	Passenger 8735	12:36	Passenger 9758
35	0:28	Passenger 368	1:36	Passenger 528	2:40	Passenger 642	3:34	Passenger 750	4:25	Passenger 875	5:26	Passenger 1088	6:16	Passenger 1525	7:09	Passenger 2198	8:14	Passenger 3455	8:55	Passenger 5212	9:54	Passenger 6788	10:56	Passenger 7785	11:45	Passenger 8752	12:38	Passenger 9777
36	0:30	Passenger 372	1:36	Passenger 530	2:40	Passenger 643	3:36	Passenger 752	4:26	Passenger 877	5:28	Passenger 1095	6:18	Passenger 1535	7:11	Passenger 2220	8:16	Passenger 3476	8:57	Passenger 5243	9:55	Passenger 6815	10:56	Passenger 7802	11:45	Passenger 8768	12:38	Passenger 9795
37	0:32	Passenger 375	1:38	Passenger 533	2:40	Passenger 645	3:36	Passenger 754	4:27	Passenger 879	5:30	Passenger 1101	6:19	Passenger 1545	7:11	Passenger 2242	8:16	Passenger 3498	8:57	Passenger 5275	9:56	Passenger 6842	10:58	Passenger 7818	11:45	Passenger 8785	12:39	Passenger 9813
38	0:34	Passenger 378	1:40	Passenger 535	2:40	Passenger 647	3:38	Passenger 756	4:27	Passenger 882	5:30	Passenger 1108	6:19	Passenger 1555	7:13	Passenger 2263	8:17	Passenger 3520	8:57	Passenger 5307	9:56	Passenger 6868	10:58	Passenger 7835	11:46	Passenger 8802	12:41	Passenger 9832
39	0:34	Passenger 382	1:42	Passenger 538	2:42	Passenger 648	3:39	Passenger 758	4:29	Passenger 884	5:30	Passenger 1115	6:20	Passenger 1565	7:14	Passenger 2285	8:17	Passenger 3541	8:57	Passenger 5338	9:57	Passenger 6885	11:00	Passenger 7852	11:47	Passenger 8818	12:41	Passenger 9850
40	0:34	Passenger 385	1:43	Passenger 540	2:44	Passenger 650	3:41	Passenger 760	4:29	Passenger 886	5:31	Passenger 1122	6:20	Passenger 1575	7:14	Passenger 2307	8:17	Passenger 3563	8:58	Passenger 5370	9:58	Passenger 6902	11:01	Passenger 7868	11:48	Passenger 8835	12:42	Passenger 9868
41	0:35	Passenger 388	1:43	Passenger 543	2:44	Passenger 652	3:41	Passenger 762	4:30	Passenger 889	5:31	Passenger 1128	6:21	Passenger 1585	7:15	Passenger 2328	8:19	Passenger 3585	8:59	Passenger 5402	9:58	Passenger 6918	11:02	Passenger 7885	11:48	Passenger 8852	12:44	Passenger 9887
42	0:37	Passenger 392	1:43	Passenger 545	2:45	Passenger 653	3:41	Passenger 764	4:31	Passenger 891	5:31	Passenger 1135	6:23	Passenger 1595	7:17	Passenger 2350	8:19	Passenger 3606	8:59	Passenger 5428	10:00	Passenger 6935	11:03	Passenger 7902	11:48	Passenger 8868	12:44	Passenger 9905
43	0:37	Passenger 395	1:45	Passenger 548	2:47	Passenger 655	3:41	Passenger 766	4:32	Passenger 893	5:33	Passenger 1142	6:24	Passenger 1605	7:18	Passenger 2372	8:20	Passenger 3628	8:59	Passenger 5455	10:01	Passenger 6952	11:03	Passenger 7918	11:48	Passenger 8885	12:45	Passenger 9923
44	0:39	Passenger 398	1:46	Passenger 550	2:48	Passenger 657	3:41	Passenger 768	4:34	Passenger 896	5:34	Passenger 1148	6:26	Passenger 1615	7:20	Passenger 2393	8:22	Passenger 3660	9:00	Passenger 5482	10:02	Passenger 6968	11:03	Passenger 7935	11:49	Passenger 8902	12:46	Passenger 9942
45	0:41	Passenger 402	1:47	Passenger 553	2:49	Passenger 658	3:42	Passenger 770	4:36	Passenger 898	5:34	Passenger 1155	6:26	Passenger 1625	7:22	Passenger 2415	8:22	Passenger 3691	9:00	Passenger 5508	10:04	Passenger 6985	11:03	Passenger 7952	11:49	Passenger 8918	12:48	Passenger 9960
46	0:43	Passenger 405	1:48	Passenger 555	2:51	Passenger 660	3:44	Passenger 772	4:36	Passenger 900	5:34	Passenger 1162	6:28	Passenger 1635	7:22	Passenger 2437	8:23	Passenger 3723	9:00	Passenger 5535	10:06	Passenger 7002	11:04	Passenger 7968	11:51	Passenger 8935	12:50	Passenger 9978
47	0:43	Passenger 408	1:48	Passenger 558	2:52	Passenger 662	3:46	Passenger 774	4:38	Passenger 903	5:36	Passenger 1168	6:29	Passenger 1645	7:22	Passenger 2458	8:23	Passenger 3755	9:00	Passenger 5562	10:07	Passenger 7018	11:04	Passenger 7985	11:53	Passenger 8952	12:50	Passenger 9997
48	0:43	Passenger 412	1:50	Passenger 560	2:54	Passenger 663	3:47	Passenger 776	4:39	Passenger 905	5:38	Passenger 1175	6:31	Passenger 1655	7:22	Passenger 2480	8:23	Passenger 3786	9:02	Passenger 5588	10:09	Passenger 7035	11:04	Passenger 8002	11:55	Passenger 8968	12:51	Passenger 10015
49	0:43	Passenger 415	1:52	Passenger 563	2:54	Passenger 665	3:49	Passenger 778	4:40	Passenger 907	5:39	Passenger 1182	6:31	Passenger 1665	7:22	Passenger 2502	8:23	Passenger 3818	9:02	Passenger 5615	10:09	Passenger 7052	11:05	Passenger 8018	11:56	Passenger 8985	12:51	Passenger 10033
50	0:44	Passenger 418	1:54	Passenger 565	2:55	Passenger 667	3:50	Passenger 780	4:42	Passenger 910	5:39	Passenger 1188	6:32	Passenger 1675	7:24	Passenger 2523	8:23	Passenger 3850	9:04	Passenger 5642	10:11	Passenger 7068	11:06	Passenger 8035	11:57	Passenger 9002	12:51	Passenger 10052
51	0:46	Passenger 422	1:56	Passenger 568	2:57	Passenger 668	3:50	Passenger 782	4:42	Passenger 912	5:41	Passenger 1195	6:34	Passenger 1685	7:26	Passenger 2545	8:23	Passenger 3881	9:06	Passenger 5668	10:13	Passenger 7085	11:06	Passenger 8052	11:58	Passenger 9018	12:52	Passenger 10070
52	0:47	Passenger 425	1:58	Passenger 570	2:57	Passenger 670	3:52	Passenger 784	4:44	Passenger 914	5:42	Passenger 1202	6:34	Passenger 1695	7:27	Passenger 2566	8:23	Passenger 3913	9:08	Passenger 5695	10:13	Passenger 7102	11:08	Passenger 8068	11:59	Passenger 9035	12:53	Passenger 10088
53	0:49	Passenger 428	2:00	Passenger 573	2:57	Passenger 672	3:53	Passenger 786	4:45	Passenger 917	5:44	Passenger 1208	6:35	Passenger 1705	7:29	Passenger 2588	8:24	Passenger 3945	9:08	Passenger 5722	10:14	Passenger 7118	11:10	Passenger 8085	12:00	Passenger 9052	12:54	Passenger 10107
54	0:51	Passenger 430	2:02	Passenger 575	2:59	Passenger 673	3:53	Passenger 788	4:47	Passenger 919	5:45	Passenger 1215	6:35	Passenger 1715	7:29	Passenger 2610	8:24	Passenger 3976	9:09	Passenger 5748	10:14	Passenger 7135	11:10	Passenger 8102	12:01	Passenger 9068	12:54	Passenger 10125
55	0:51	Passenger 433	2:04	Passenger 578	2:59	Passenger 675	3:55	Passenger 790	4:47	Passenger 921	5:45	Passenger 1222	6:36	Passenger 1725	7:31	Passenger 2631	8:26	Passenger 4008	9:10	Passenger 5775	10:14	Passenger 7152	11:11	Passenger 8118	12:01	Passenger 9085	12:56	Passenger 10145
56	0:52	Passenger 435	2:05	Passenger 580	2:59	Passenger 677	3:57	Passenger 792	4:47	Passenger 924	5:46	Passenger 1228	6:36	Passenger 1735	7:33	Passenger 2653	8:26	Passenger 4040	9:11	Passenger 5802	10:15	Passenger 7168	11:12	Passenger 8135	12:01	Passenger 9102	12:56	Passenger 10165
57	0:54	Passenger 438	2:07	Passenger 582	3:00	Passenger 678	3:59	Passenger 794	4:47	Passenger 926	5:47	Passenger 1235	6:38	Passenger 1745	7:34	Passenger 2675	8:26	Passenger 4071	9:13	Passenger 5828	10:17	Passenger 7185	11:14	Passenger 8152	12:03	Passenger 9118	12:57	Passenger 10185
58	0:56	Passenger 440	2:09	Passenger 583	3:01	Passenger 680	3:59	Passenger 796	4:47	Passenger 928	5:48	Passenger 1242	6:40	Passenger 1755	7:35	Passenger 2696	8:28	Passenger 4103	9:14	Passenger 5855	10:19	Passenger 7202	11:15	Passenger 8168	12:04	Passenger 9135	12:59	Passenger 10205

Pedestrian simulation in UnityC#

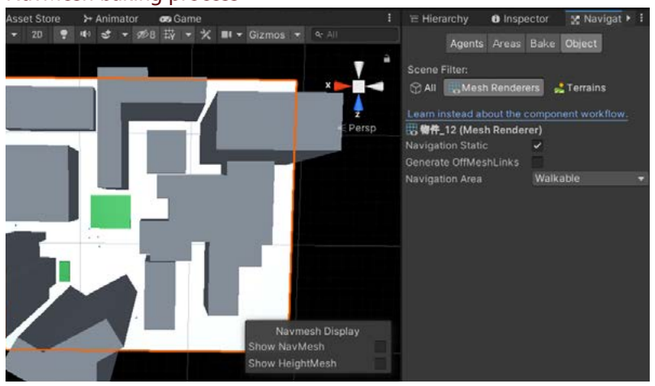
An agent-based model (ABM) is a class of computational models for simulating the actions and interactions of autonomous agents (both individual or collective entities such as organizations or groups) with a view to assessing their effects on the system as a whole. It combines elements of game theory, complex systems, emergence, computational sociology, multi-agent systems, and evolutionary programming.



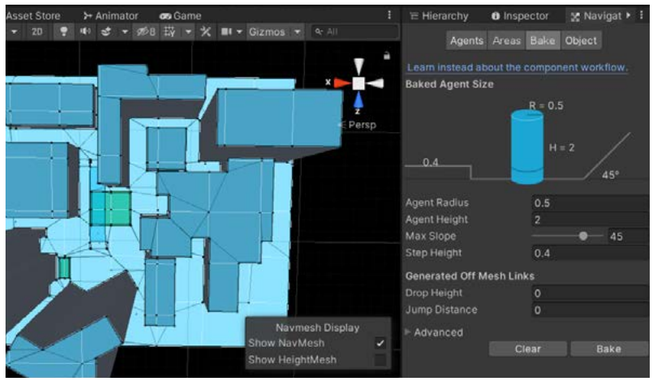
Navmesh in Unity

NavMesh (short for Navigation Mesh) is a data structure which describes the walkable surfaces of the game world and allows to find path from one walkable location to another in the game world. The data structure is built, or baked, automatically from the level geometry.

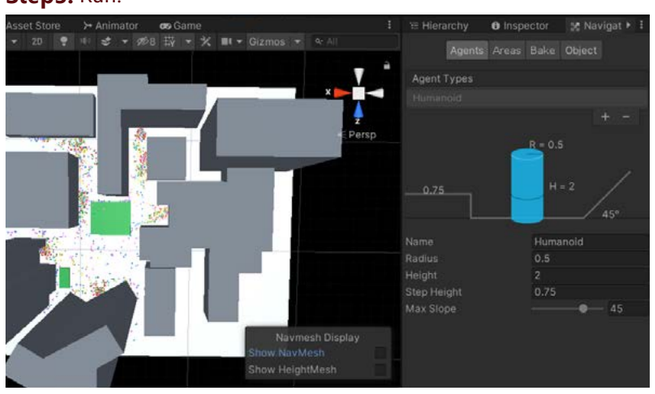
Step1: Check Navigation Static on to include selected objects in the NavMesh baking process



Step2: Adjust the bake settings and bake the navmesh.



Step3: Run!



NavMesh Agent component help create characters which avoid each other while moving towards their goal. Agents reason about the game world using the NavMesh and they know how to avoid each other as well as moving obstacles.

NavMesh Obstacle component allows to describe moving obstacles the agents should avoid while navigating the world.
While the obstacle is moving the agents do their best to avoid it, but once the obstacle becomes stationary it will carve a hole in the navmesh so that the agents can change their paths to steer around it, or if the stationary obstacle is blocking the path way, the agents can find a different route.

Define the "allController" class. This class aims at defining and generating new agents.

```
5  public class allController : MonoBehaviour
```

STEP1: Assign attributes

```
7      private List<GameObject> SpawnLocationlist = new List<GameObject>();
8      private List<GameObject> AgentTypeslist = new List<GameObject>();
9      private int AgentTypesCount;
10     private int SpawnLocationType;
11     public int AgentAllCount = 60000;
```

STEP2: AgentTypes

"GetAgentType" Method is called before the first frame update. Find and count all agents.

```
25     void GetAgentType()
26     {
27         GameObject parentobject = GameObject.Find("AgentTypes");
28         AgentTypesCount = parentobject.transform.childCount;
29
30         for (int i = 0; i < AgentTypesCount; i++)
31         {
32             GameObject AgentType = parentobject.transform.GetChild(i).gameObject;
33             AgentTypeslist.Add(AgentType);
34         }
35     }
```

STEP3: SpawnLocation

"GetSpawnLocation" Method is called before the first frame update. Find and count all the location where agents are generated.

```
36     void GetSpawnLocation()
37     {
38         GameObject parentobject = GameObject.Find("SpawnLocation");
39         SpawnLocationType = parentobject.transform.childCount;
40
41         for (int i = 0; i< SpawnLocationType; i++)
42         {
43             GameObject SpawnLocation = parentobject.transform.GetChild(i).gameObject;
44             SpawnLocationlist.Add(SpawnLocation);
45         }
46     }
```

STEP4:

"SpawnAgents" Method is called once per frame. Generate new agents until the total number of agents reach 60000("AgentAllCount")

```
47     void SpawnAgents()
48     {
49         int i = Random.Range(0, AgentTypesCount);
50         GameObject AgentSelected = AgentTypeslist[i];
51
52         int m = Random.Range(0, SpawnLocationType);
53         GameObject SpawnLocationSelected = SpawnLocationlist[m];
54
55         if (AgentTypeslist.Count<= AgentAllCount)
56         {
57             GameObject AgenttoSpawn = Instantiate(AgentSelected,
                   SpawnLocationSelected.transform.position, SpawnLocationSelected.transform.rotation);
58             AgentTypeslist.Add(AgenttoSpawn);
59         }
60     }
```

Define the "Controller" class. This class aims at defining targets and enabling agents move towards targets.

```
7  public class Controller : MonoBehaviour
```

STEP1: Assign attributes

```
9      private NavMeshAgent meshAgent;
10     private List<GameObject> alltarget = new List<GameObject>();
11     private List<GameObject> TargetTypeslist = new List<GameObject>();
12     private float distance;
13     private int rand;
14     private int TargetCount;
15     private GameObject destination;
16     private bool moveAgain = false;
```

STEP2: Target

"PinkTarget" Method is called before the first frame update. Find and count all targets.

```
25     void PinkTarget()
26     {
27         GameObject parentobject = GameObject.Find("Target");
28         TargetCount = parentobject.transform.childCount;
29
30         for (int i = 0; i < TargetCount; i++)
31         {
32             GameObject TargetType = parentobject.transform.GetChild(i).gameObject;
33             TargetTypeslist.Add(TargetType);
34         }
35     }
```

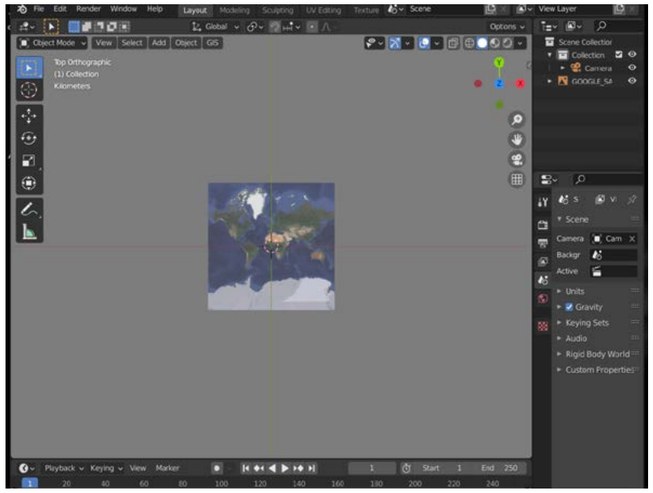
STEP2: "Update" Method is called per frame. Agents move towards targets. When the distance is less than 5, agents stop moving.

```
37     void Update()
38     {
39         if (destination != null)
40         {
41             distance = Vector3.Distance(gameObject.transform.position,
                 destination.transform.position);
42         }
43         if (distance < 5)
44         {
45             moveAgain = true;
46         }
47         if(moveAgain == true)
48         {
49             rand = UnityEngine.Random.Range(0, TargetCount);
50             destination = TargetTypeslist[rand];
51             meshAgent.SetDestination(destination.transform.position);
52             Debug.Log("Rand" + rand);
53             moveAgain = false;
54         }
55     }
```

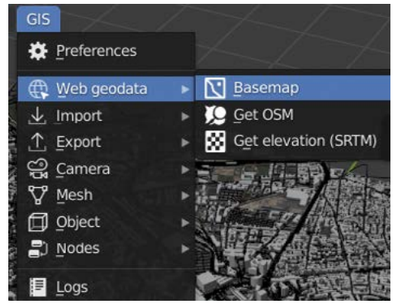
Agent behavior simulation in a plaza of Ladefense

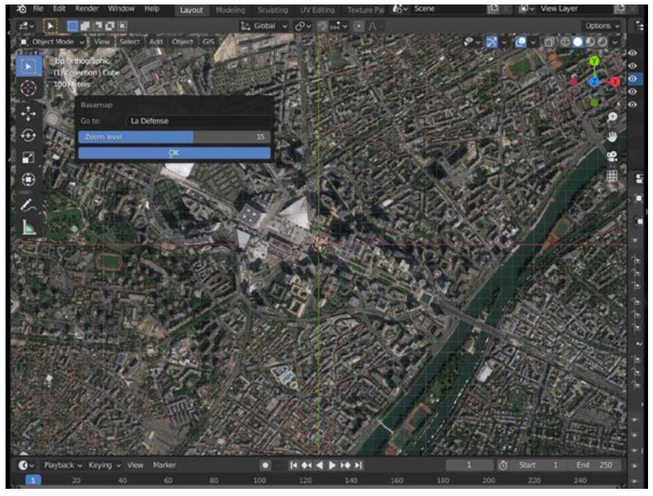
Urban-Scale Modelling based on Blender&GIS pluggin

Grab geodata directly from the web : display dynamics web maps inside Blender 3d view, requests for OpenStreetMap data (buildings, roads ...), get true elevation data from the NASA SRTM mission.

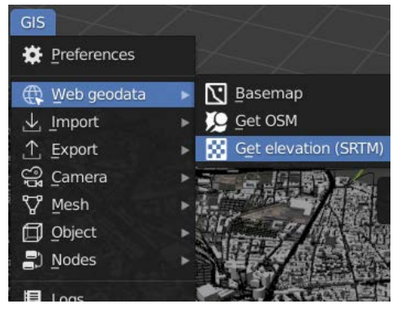


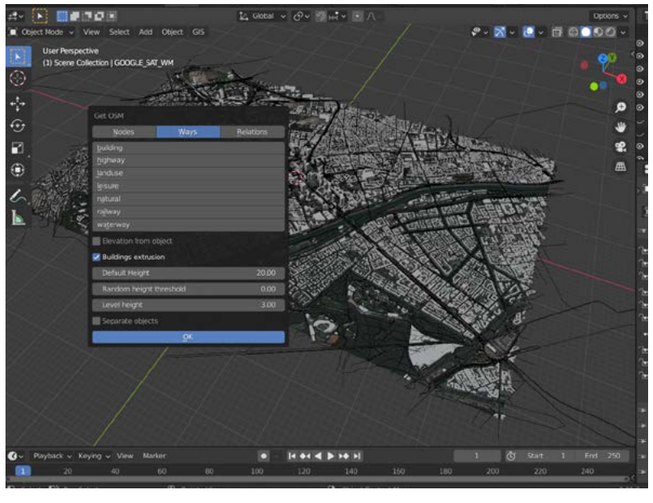
STEP1:
Display dynamics web maps inside Blender 3d view by GIS pluggin



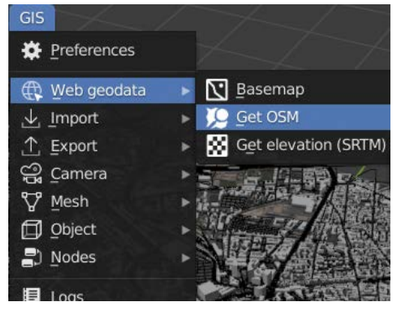


STEP2:
Choose the geographical position
Zoom the web maps according to the required scale and accuracy





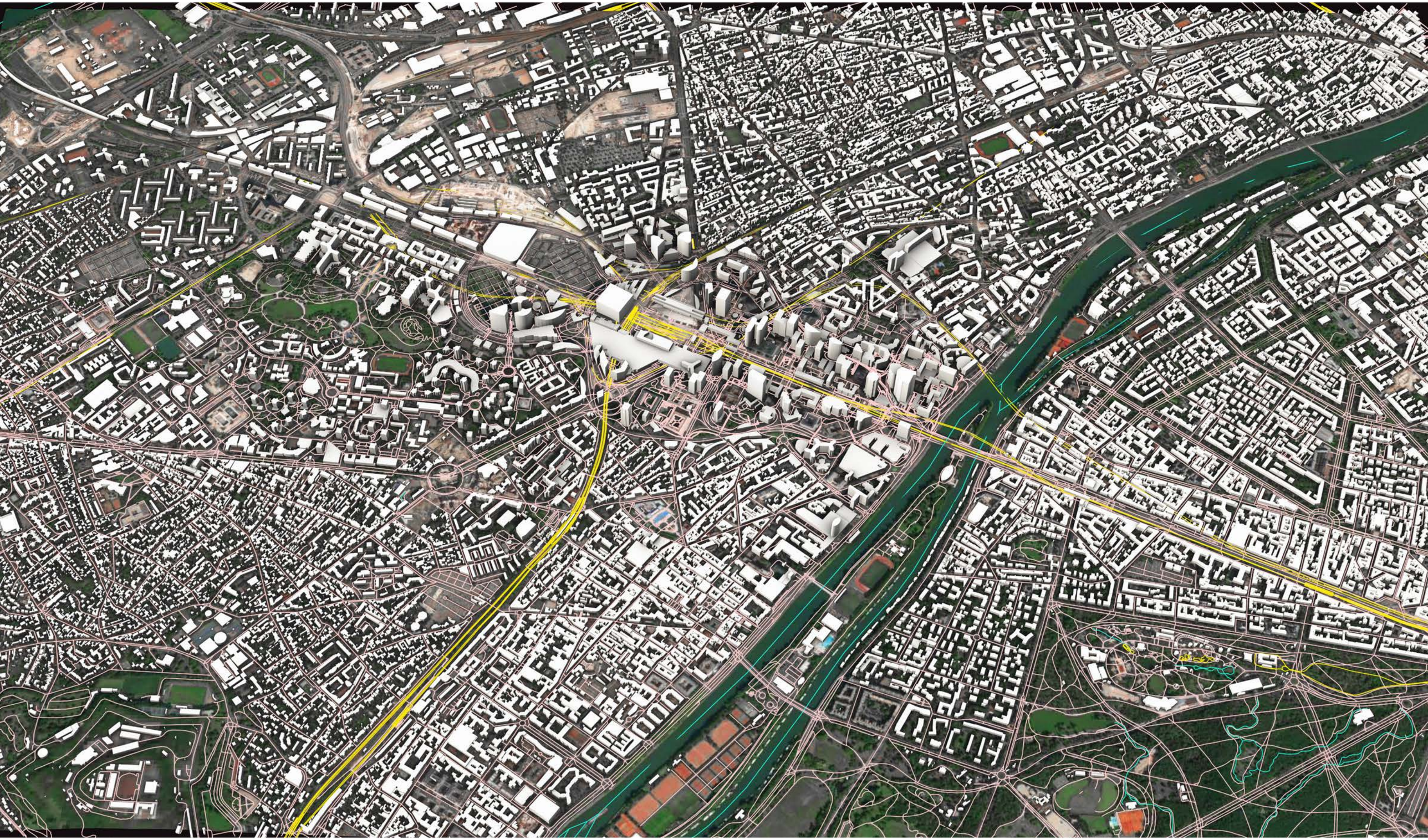
STEP3:
requests for OpenStreetMap data (buildings, roads ...), get true elevation data from the NASA SRTM mission



RELATIVE TUTORIAL: https://github.com/domlysz/BlenderGIS.git

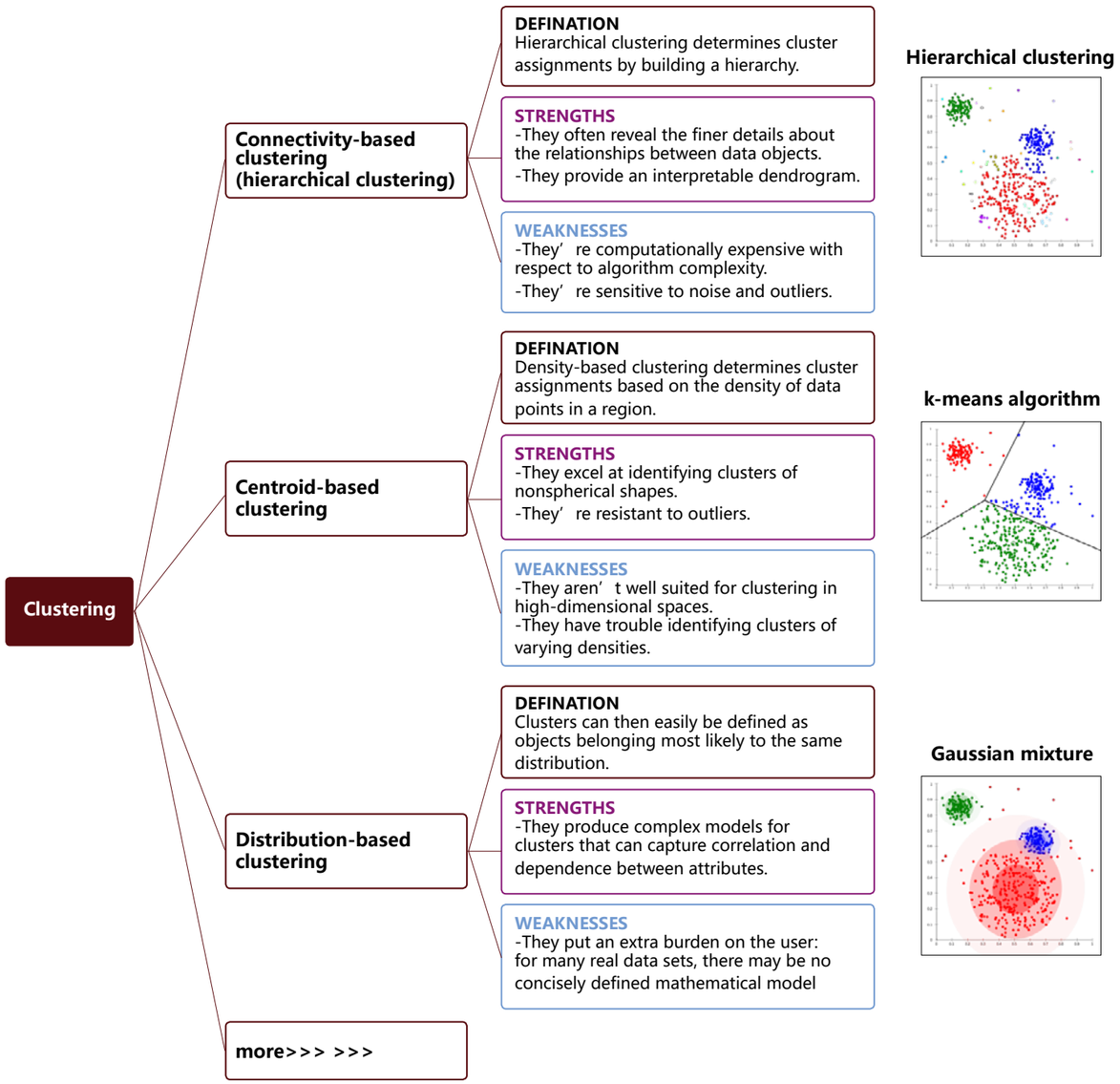
Ladefense urban-scale model within the radius of 1.5 km

RELATIVE TUTORIAL: https://github.com/domlysz/BlenderGIS.git



Clustering analysis

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).



Kmeans Clustering

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

Algorithm 1 k-means algorithm

1: Specify the number k of clusters to assign.
2: Randomly initialize k centroids.
3: **repeat**
4: **expectation:** Assign each point to its closest centroid.
5: **maximization:** Compute the new centroid (mean) of each cluster.
6: **until** The centroid positions do not change.

The new function centroid is based on the tagged photo position of Flicker,where people meet and record most. kmeans is used to reorganize and relocate the function distribution based on the social media data.

STEP1: Import csv

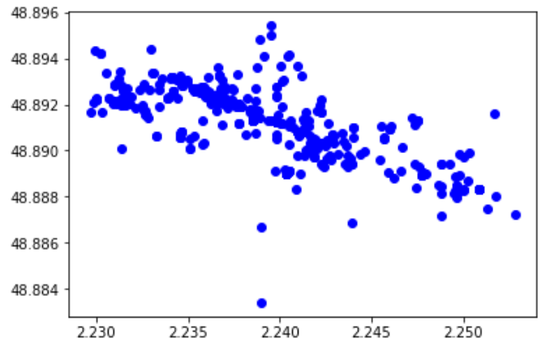
```
data = pd.read_csv('results1-office.csv')
data.head()
```

	farm	latitude	longitude
0	66	48.889574	2.242777
1	66	48.890086	2.231369
2	66	48.889922	2.250294
3	66	48.892301	2.238204
4	66	48.893192	2.236088

id	owner	secret	server	title	realname	tags	machine_t	latitude	longitude	woeid	media	media_sta
5.07E+10	99935530(	edba0b00	65535	Business C	Alexander	city urban architectu		48.88957	2.242777	8504417	photo	ready
4.9E+10	15069464(	fcf44ff650	65535	From my I	Patrice LIE	panorama horizon h		48.88999	2.241564	8504417	photo	ready
4.8E+10	10016724(	607f13b3a	65535	Abstract L	Martijn J	building hirise office		48.88985	2.244408	55863561	photo	ready
4.04E+10	96956876(	c49387cd	820	Paris_La_E	Ivan Sgual	architecture architet		48.88944	2.243911	55863561	photo	ready
3.26E+10	40121274(	756f7952b	383	Office Hal	Claude Fa	lines hall noiretblanc		48.89009	2.241581	12523323	photo	ready
1.6E+10	63488389(	877298a5	8586	B芒timent	Friendy W	city windows urban		48.89064	2.242949	55863561	photo	ready
1.47E+10	99616828(	497c0734(	5567	Multi Glas	Philippe S	city urban abstract n		48.8894	2.243549	55863561	photo	ready
1.43E+10	63234672(	b9ae8691	3871	EDF Towe	Mustang	paris france building		48.89011	2.242005	55863561	photo	ready
1.11E+10	52198061(	2d4701c7(	3692	My desk		paris la of flickriosap		48.88914	2.240514	22657641	photo	ready
1.01E+10	52198061(	4f572b55f	7304	I Novell		office nov flickriosap		48.88914	2.240514	22657641	photo	ready
9.78E+09	94735786(	9497fc060	3781	La Jet矛e	David Mcl	paris france europe /		48.88974	2.241841	12523323	photo	ready
8.53E+09	99616828(	86e7af5f3(	8097	City Place	Philippe S	street city urban mo		48.89054	2.24187	22657641	photo	ready
8.3E+09	99616828(	885a421a5	8074	This is The	Philippe S	street city light sunse		48.89046	2.241924	22657641	photo	ready
8.06E+09	82551583(	05f520162	8322	Tour EDF		france skyscraper off		48.89034	2.24164	55863561	photo	ready
6.06E+09	52574705(	c938c30d6	6208	SUSE Architects		summer me fun offic		48.889	2.240363	22657641	photo	ready
6.06E+09	52574705(	63554bb3:	6182			summer fun office w		48.889	2.240363	22657641	photo	ready
5.26E+09	37611009(	9b2eed59	5127	that's wha	Pierre Site	city paris france tow		48.88937	2.242316	22657641	photo	ready

STEP2: Visualise data points

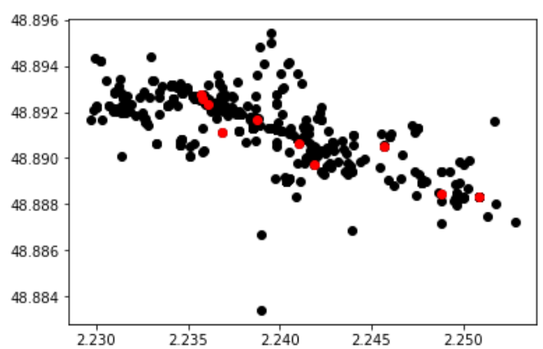
```
X = data[["longitude","latitude"]]
plt.scatter(X["longitude"],X["latitude"],c='blue')
plt.show()
```



STEP3: Specify the number of K(Next page will show how to specify K)
Randomly intialize k centroids

```
K=10
Centroids = (X.sample(n=K))
plt.scatter(X["longitude"],X["latitude"],c='black')
plt.scatter(Centroids["longitude"],Centroids["latitude"],c='red')
```

```
<matplotlib.collections.PathCollection at 0x287c2152fd0>
```



STEP4: Assign all the points to the closest cluster centroid

```
diff = 1
j=0

while(diff!=0):
    XD=X
    i=1
    for index1,row_c in Centroids.iterrows():
        ED=[]
        for index2,row_d in XD.iterrows():
            d1=(row_c["longitude"]-row_d["longitude"])**2
            d2=(row_c["latitude"]-row_d["latitude"])**2
            d=np.sqrt(d1+d2)
            ED.append(d)
        X[i]=ED
        i=i+1
```

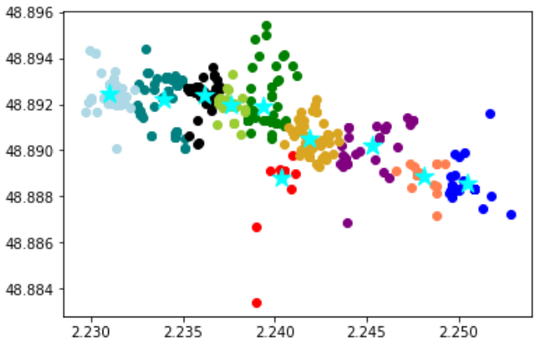
STEP5: Recompute centroids of newly formed clusters

```
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos=i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new = X.groupby(["Cluster"]).mean()[["latitude","longitude"]]
    if j == 0:
        diff=1
        j=j+1
    else:
        diff = (Centroids_new['latitude'] - Centroids['latitude']).sum() + (Centroids_new['longitude'] - Cen
    Centroids = X.groupby(["Cluster"]).mean()[["latitude","longitude"]]
```

STEP6: Final clustering and centroids

```
color=['purple','green','red','teal','black','yellowgreen','blue','coral','goldenrod','lightblue']
for k in range(K):
    data=X[X["Cluster"]==k+1]
    plt.scatter(data["longitude"],data["latitude"],c=color[k])
plt.scatter(Centroids["longitude"],Centroids["latitude"],c='cyan',marker='*',s=200)
plt.figure(figsize=(6, 15))
print(Centroids)
```

Cluster	latitude	longitude
1	48.890234	2.245306
2	48.891927	2.239340
3	48.888823	2.240342
4	48.892202	2.233941
5	48.892394	2.236183
6	48.891960	2.237616
7	48.888584	2.250480
8	48.888855	2.248086
9	48.890523	2.241904
10	48.892442	2.231031



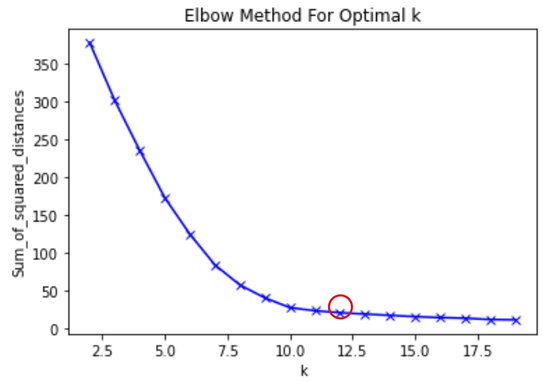
Elbow Method For Optimal k

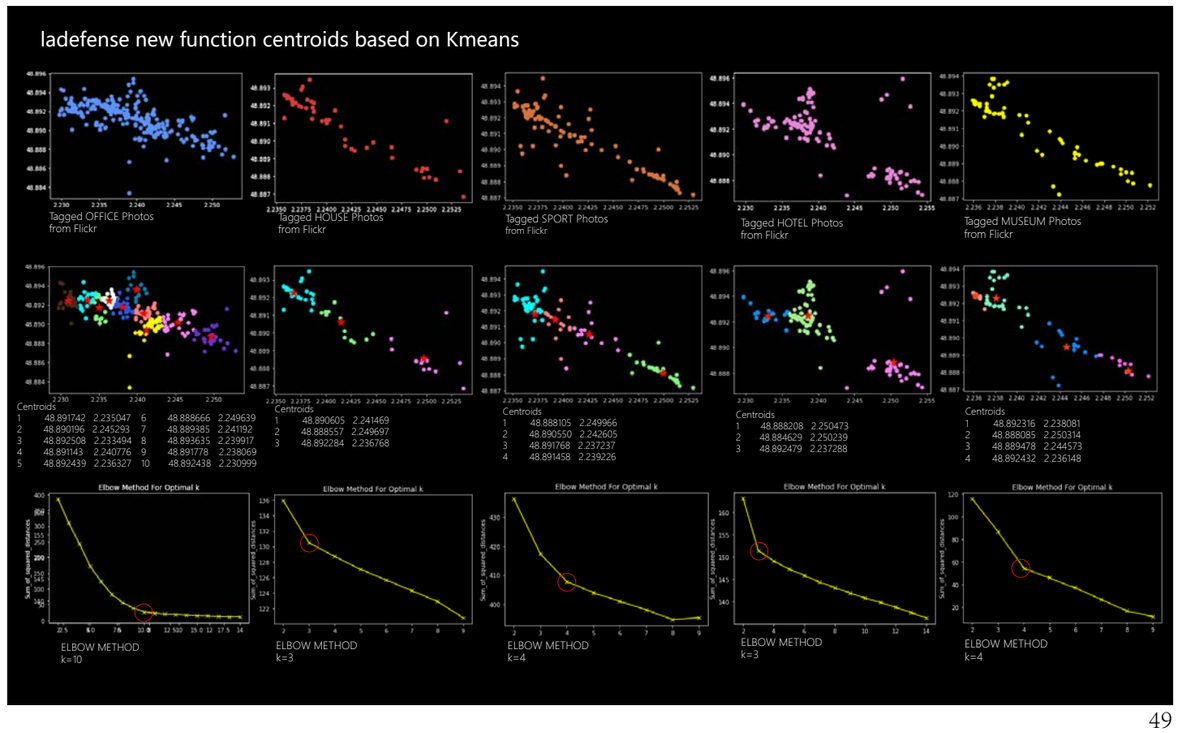
Elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use.

```
mms = MinMaxScaler()
mms.fit(data)
data_transformed = mms.transform(data)
```

```
Sum_of_squared_distances = []
K = range(2,20)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(data_transformed)
    Sum_of_squared_distances.append(km.inertia_)
```

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```





3D-Kmeans

STEP1: Import csv

```
data = pd.read_csv('cluster1-pop.csv')
```

```
X = data[["xcor","ycor","zcor"]]
Y = data[["xcor","ycor"]]
x=np.array(X["xcor"],dtype=float)
y=np.array(X["ycor"],dtype=float)
z=np.array(X["zcor"],dtype=float)
```

STEP2: 3D kmeans implementation

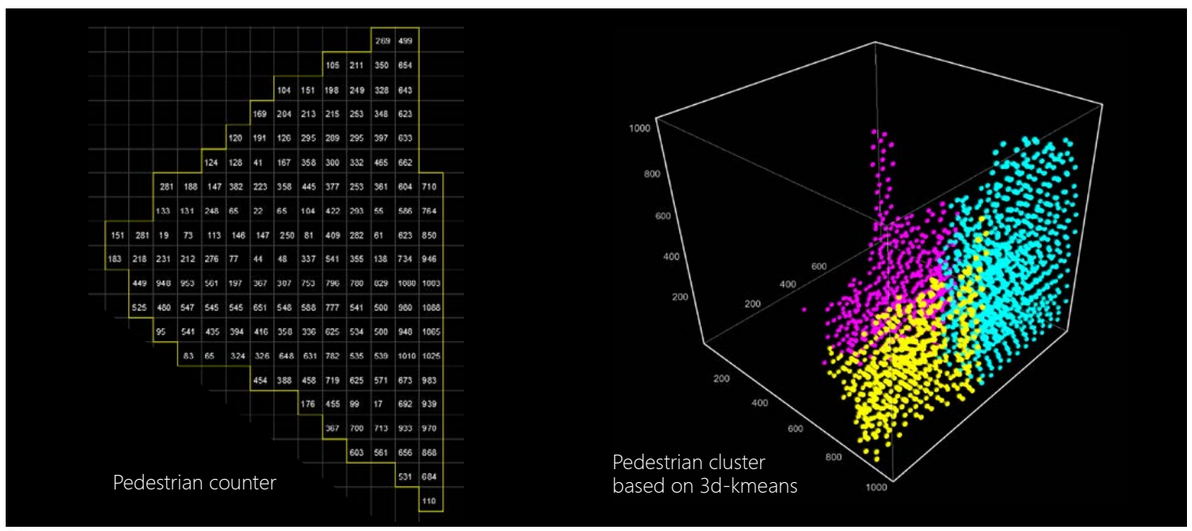
```
fig = ipv.figure(height=600, width=600, layout={'width':'50%', 'height':'50%'})
scatter = ipv.scatter(x,y,z, size=2, marker="sphere",color="red",opacity=0.03, selection=None)
```

```
def hex_to_rgb(hex):
    hex = hex[1:]
    return struct.unpack('BBB', bytes.fromhex(hex))
```

```
def handle_cp_change(labels, **groups):
    group_ids = [int(g.split(' ')[1]) for g in groups.keys()]
    group_color = {k: hex_to_rgb(get_cp_value(cp)) for k, cp in zip(group_ids, groups.values())}
    colors = list(map(lambda x: group_color[x], labels))
    scatter.color = colors

def get_cp_value(cp):
    if type(cp) == ColorPicker:
        return cp.value
    else:
        return cp
```

```
avaliable_colors = {
    0: '#ff0000',
    1: '#00ff00',
    2: '#0000ff',
}
```

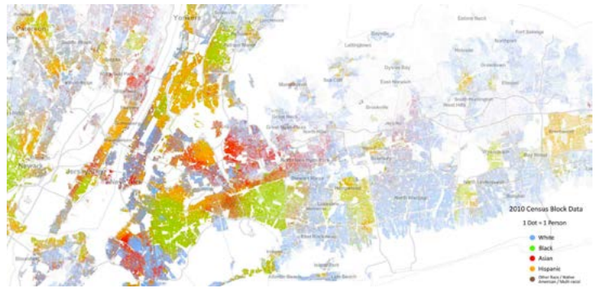
```
def color_scatter_with_kmeans(n_clusters):
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans = kmeans.fit(Y)
    labels = kmeans.predict(Y)
    color_pickers = {f'group {k}': ColorPicker(value=avaliable_colors[k%len(avaliable_colors)], description=
                     for k in range(n_clusters)}
    handle_cp_change(labels=list(labels), **color_pickers)
    return interact(handle_cp_change, labels=fixed(list(labels)) , **color_pickers)
```



Schelling Segregation Model

The Schelling model of segregation is an agent-based model that illustrates how individual tendencies regarding neighbors can lead to segregation. The model is especially useful for the study of residential segregation of ethnic groups where agents represent householders who relocate in the city.

In the model, each agent belongs to one of two groups and aims to reside within a neighborhood where the fraction of 'friends' is sufficiently high: above a predefined tolerance threshold value F. It is known that depending on F, for groups of equal size, Schelling's residential pattern converges to either complete integration (a random-like pattern) or segregation.



The New York City map represents data from the 2010 US Census color-coded by race.

Theory of model

Step1

Suppose there are two types of agents: X and O. The two types of agents might represent different races, ethnicity, economic status, etc. Two populations of the two agent types are initially placed into random locations of a neighborhood represented by a grid. After placing all the agents in the grid, each cell is either occupied by an agent or is empty.

Agents placed randomly in grid

X	X	O	X	O
	O	O	O	O
X	X			
X	O	X	X	X
X	O	O		O

Step2

An important step is to define if each agent is satisfied with its current location.
A satisfied agent is one that is surrounded by at least t percent of agents that are like itself. This threshold t is one that will apply to all agents in the model, even though in reality everyone might have a different threshold they are satisfied with.
Note that the higher the threshold, the higher the likelihood the agents will not be satisfied with their current location.

Satisfied because **1/2** (50%) of neighbors are X

X	X	O	X	O
	O	O	O	O
X	X			
X	O	X	X	X
X	O	O		O

Dissatisfied because only 1/4 (25%) of neighbors are X

X	X	O	X	O
	O	O	O	O
X	X			
X	O	X	X	X
X	O	O		O

Step3

When an agent is not satisfied, it can be moved to any vacant location in the grid. Any algorithm can be used to choose this new location. For example, a randomly selected cell may be chosen, or the agent could move to the nearest available location.

Dissatisfied agents marked with *

X	X*	O	X*	O
	O	O	O	O
X	X			
X	O*	X	X	X
X	O	O		O*

All dissatisfied agents relocated

X		O		O
O	O	O	O	O
X	X	X		X
X		X	X	X
X	O	O	O	

STEP1: Define the agent class with various parameters(kind, preference of the neighbours and the world environment). Define its neighbours, the preferred neighbours, its happiness and dissatisficory towards its neighbours(The more similar neighbours the agent have, the happier it is) and the vacany.

```
class Agent:
    def __init__(self, x, y, kind, preference, world):
        self.x = x
        self.y = y
        self.kind = kind
        self.preference = preference
        self.world = world

    def know_neighbours(self):
        self.neighbours = [self.world.coords[x % self.world.size, y % self.world.size]
                           for x, y in itertools.product(range(self.x-1, self.x+2),
                                                         range(self.y-1, self.y+2))]

    def swap(self, partner):
        self.kind, partner.kind = partner.kind, self.kind

    def happiness(self):
        neighbour_kinds = [n.kind for n in self.neighbours]
        numsame = neighbour_kinds.count(self.kind)
        numvacant = neighbour_kinds.count(0)
        if numvacant == 8:
            if self.vacant():
                return 1.0
            return 0.0
        return (numsame - 1) / (8 - numvacant)

    def dissatisfied(self):
        return self.happiness() < self.preference

    def vacant(self):
        return self.kind == 0
```

STEP2: Define the world class with various parameters. A grid of squares represents the world, with each square representing a plot of land in which an agent can live.The world is populated (randomly initially) with a number of different kinds of agents, leaving a proportion of the squares vacant.

```
class World:
    def __init__(self, size, kinds, probs, preference):
        self.size = size
        self.preference = preference
        self.coords = self.populate_world(kinds, probs, preference)
        [a.know_neighbours() for row in self.coords for a in row]
        self.num_dissatisfied = size ** 2
        self.num_vacant = sum([a.vacant() for row in self.coords for a in row])

    def populate_world(self, kinds, probs, preference):
        return np.array([[Agent(x, y, np.random.choice(kinds, p=probs), preference, self) for y in range(se
```

```
    def advance_turn(self):
        dissatisfied = []
        vacant = []
        for row in self.coords:
            for a in row:
                if a.vacant():
                    vacant.append(a)
                elif a.dissatisfied():
                    dissatisfied.append(a)
        self.num_dissatisfied = len(dissatisfied)
        np.random.shuffle(dissatisfied)
        np.random.shuffle(vacant)
        num_swap = min(self.num_dissatisfied, self.num_vacant)
        for indx in range(num_swap):
            dissatisfied[indx].swap(vacant[indx])
```


STEP3: Another measure of segregation is the number of ‘colonies’ formed. That is how many clusters of similar agents are formed. For the purpose of illustration, I will very loosely define a colony as follows: Imagine the grid of squares as a network of vertices, where there is an edge between two vertices if those squares are neighbours and they are occupied by agents of the same kind. Then a colony is a connected component of that network.

```
def number_connected_components(w):
    temp = np.matrix(copy.deepcopy(w.atlas()))
    nodes = ['(' + str(x) + ', ' + str(y) + ')' for x in range(w.size) for y in range(w.size) if temp[x, y]
    G = nx.Graph()
    G.add_nodes_from(nodes)
    for x in range(w.size):
        for y in range(w.size):
            if temp[(x+1) % w.size, y] != 0 and temp[x, y] != 0:
                if temp[(x+1) % w.size, y] == temp[x, y]:
                    G.add_edge('(' + str(x) + ', ' + str(y) + ')', '(' + str((x+1) % w.size) + ', ' + str(y)
            if temp[x, (y+1) % w.size] != 0 and temp[x, y] != 0:
                if temp[x, (y+1) % w.size] == temp[x, y]:
                    G.add_edge('(' + str(x) + ', ' + str(y) + ')', '(' + str(x) + ', ' + str((y+1) % w.size)
    return nx.number_connected_components(G)
```

STEP4: Perference experiment. Set the proportion of each agent and the vacancy.

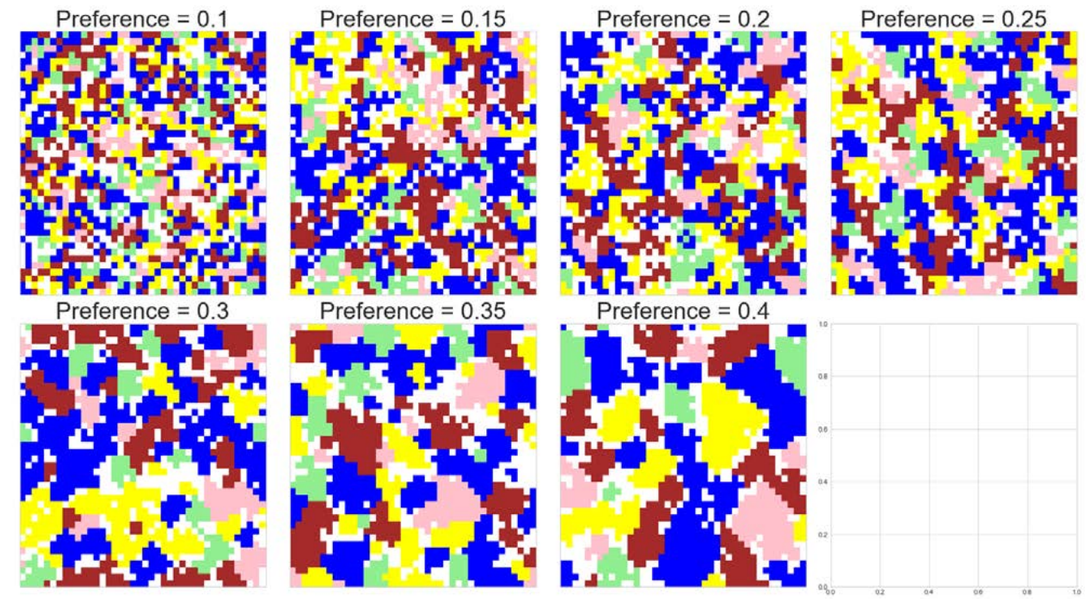
```
np.num_trials = 1
worlds = {str(round(pref, 4)): [World(40, [0, 1, 2, 3, 4, 5], [0.20, 0.15, 0.30, 0.10, 0.15, 0.10]
```

```
for k, trial in tqdm.tqdm(list(itertools.product(worlds, range(num_trials)))):
    worlds[k][trial].play(threshold=0.00005)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 43/43 [00:12<00:00,  3.53it/s]
```

```
nrows = 3
ncols = 4
cmap = colors.ListedColormap(['white', 'yellow', 'blue','pink','brown','lightgreen']) fig, axarr =
plt.subplots(nrows=nrows, ncols=ncols, figsize=(22, 18))
for i, k in enumerate(['0.1', '0.15','0.2', '0.25', '0.3', '0.35', '0.4']):
    axarr[i // ncols, i % ncols].pcolor(worlds[k][0].atlas(), cmap=cmap)
    axarr[i // ncols, i % ncols].set_xticks([])
    axarr[i // ncols, i % ncols].set_yticks([])
    axarr[i // ncols, i % ncols].set_title('Preference = ' + str(k), fontsize=34)
```

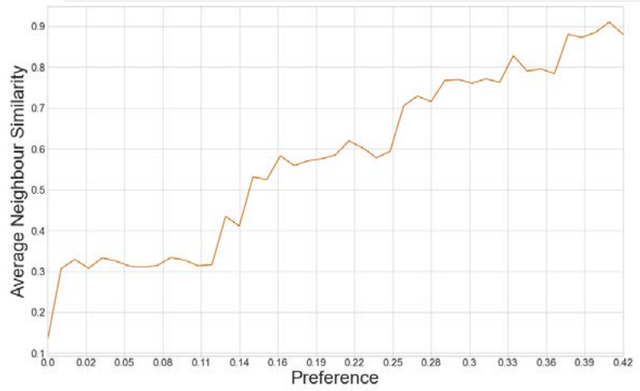
```
plt.tight_layout()
fig.savefig('increase_preference')
```



STEP5: As the individual agents' preferences are increased, the world becomes more and more segregated. We can see in the plot below that as agents' preference is increased, the resulting world's mean happiness also increases.

```
fig = plt.figure(figsize=(16, 10))
sns.tsplot([[np.mean(worlds[k][trial].happiness_distribution()) for k in sorted(worlds)] for trial in range
ticks = np.linspace(0, 42, 16)

plt.xticks(ticks, [list(sorted(worlds.keys()))[int(t)] for t in ticks])
plt.xlabel('Preference', fontsize=28)
plt.ylabel('Average Neighbour Similarity', fontsize=28)
plt.tick_params(axis='both', which='major', labelsize=16)
plt.savefig('preference_meanhappiness')
```

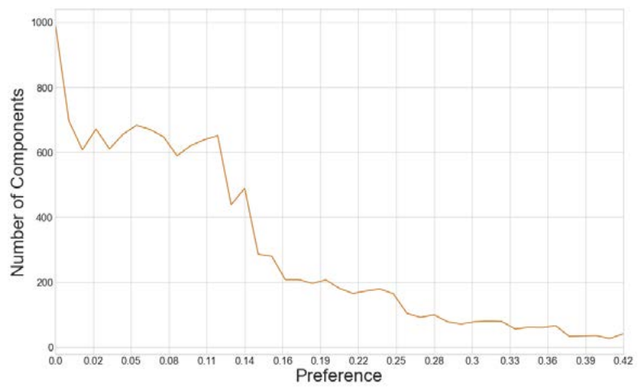
```
C:\Users\claire\miniconda3\lib\site-packages\seaborn\timeseries.py:183: UserWarning: The `tsplot` function is
deprecated and will be removed in a future release. Please update your code to use the new `lineplot` functio
n.
  warnings.warn(msg, UserWarning)
```

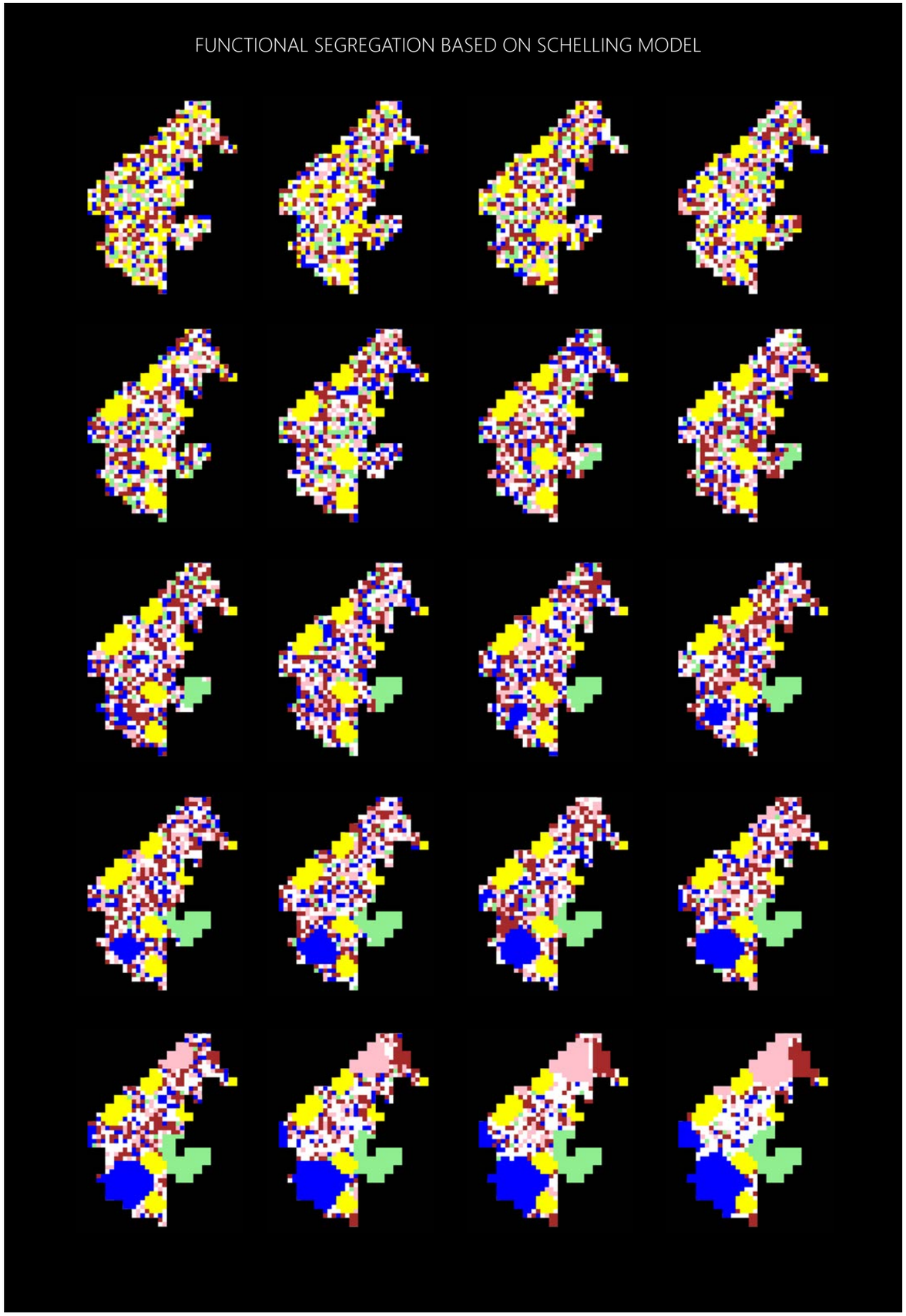


STEP6: Using NetworkX getting the number of connected components is simple. The plot below shows that the number of colonies dramatically decrease as agents' preference increases, and that the largest decrease actually occurs at a very low preference level:

```
fig = plt.figure(figsize=(16, 10))
sns.tsplot(np.transpose(components), ci=99)
ticks = np.linspace(0, 42, 16)
plt.xticks(ticks, [list(sorted(worlds.keys()))[int(t)] for t in ticks])
plt.xlabel('Preference', fontsize=28)
plt.ylabel('Number of Components', fontsize=28)
plt.tick_params(axis='both', which='major', labelsize=16)
plt.savefig('preference_components')
```

```
C:\Users\claire\miniconda3\lib\site-packages\seaborn\timeseries.py:183: UserWarning: The `tsplot` function is
deprecated and will be removed in a future release. Please update your code to use the new `lineplot` functio
n.
  warnings.warn(msg, UserWarning)
```





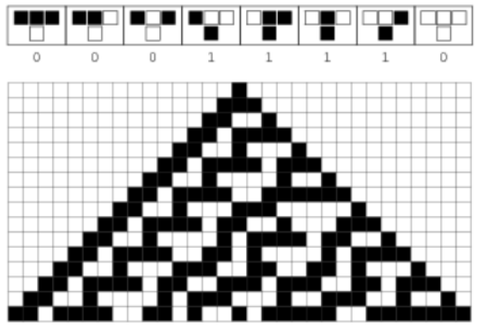
Cellular Automaton in MMA

A cellular automaton is a collection of "colored" cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells. The rules are then applied iteratively for as many time steps as desired. von Neumann was one of the first people to consider such a model, and incorporated a cellular model into his "universal constructor."

Elementary cellular automaton

The simplest nontrivial cellular automaton would be one-dimensional, with two possible states per cell, and a cell's neighbors defined as the adjacent cells on either side of it.

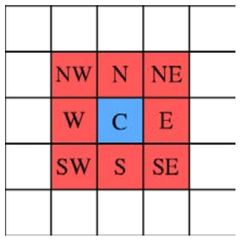
A cell and its two neighbors form a neighborhood of 3 cells, so there are 23 = 8 possible patterns for a neighborhood. A rule consists of deciding, for each pattern, whether the cell will be a 1 or a 0 in the next generation. There are then 28 = 256 possible rules.



Conway's Game of Life

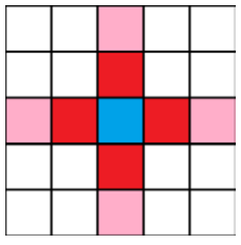
The array of cells of the automaton has two dimensions.
Each cell of the automaton has two states (conventionally referred to as "alive" and "dead", or alternatively "on" and "off")
The neighborhood of each cell is the Moore neighborhood; it consists of the eight adjacent cells to the one under consideration and (possibly) the cell itself.
In each time step of the automaton, the new state of a cell can be expressed as a function of the number of adjacent cells that are in the alive state and of the cell's own state; that is, the rule is outer totalistic (sometimes called semitotalistic).

Moore neighborhood



It is one of the two most commonly used neighborhood types, the other one being the von Neumann neighborhood. The well known Conway's Game of Life uses the Moore neighborhood. It is similar to the notion of 8-connected pixels in computer graphics.

Von Neumann neighborhood



In cellular automata, the von Neumann neighborhood (or 4-neighborhood) is classically defined on a two-dimensional square lattice and is composed of a central cell and its four adjacent cells.

Unit Aggregation Experiment

```
In[ ]:= Manipulate[
          If[bs === Sphere, o = 0];
```

evolution rule

```
With[{u = Part[
    CellularAutomaton[{rn, {2, {{2, 2, 2}, {2, 1, 2}, {1, 2, 2}}}, {1, 1}},
 {{Table[1, {init}]}, 0}, {t, All, All}], 1 + Accumulate[IntegerDigits[c, 2, t]]]},
```

generation rules:Game of Life

initial condition

generation steps

Set parameters

```
{{rn, 121268, "rule number"}, {121268, 124844, 124868}, SetterBar},
 {{c, 1, "choice number"}, 1, 2^20, 1},
 {{init, 5, "size of initial block"}, 2, 25, 1,
  Appearance -> "Labeled"},
 {{t, 6, "steps of evolution"}, 1, 15, 1,
  Appearance -> "Labeled"}, Delimiter,
 {{o, 0, "transparency"}, 0, 1,
  Enabled -> (bs === Cuboid)}, {{se, True, "show edges"}, {True, False}},
ControlPlacement -> Top]
```

Set graphics

```
With[{
   g = Graphics3D[{Opacity[1 - o],
      bs[{#2, #3, -#1}] & @@@ Position[u, 1]}, ImageSize -> 200]},
  If[Grid[{{g, Column[ArrayPlot[Sqrt[Total[#]], ImageSize -> 120] & /@ {u,
        Transpose[u], Transpose[u, {2, 3, 1}]}]}}], g]]],
```