Q4 - Microscopic Modelling

(a) The Optimal Velocity Model, is specified in eqs. 1 and 2 – referred to as Model specification 1 from now on:

$$\frac{\mathrm{d}v_\alpha}{\mathrm{d}t} = \frac{v'_e(d_\alpha) - v_\alpha}{\tau} \tag{1}$$

$$v'_\alpha(d_\alpha) = \frac{v_0}{2}\Big[\tanh(d_\alpha - d_c) + \tanh(d_c)\Big] \tag{2}$$

Consider an alternative specification – referred to as Model specification 2, specified by eqs. 3 – 5:

$$\frac{\mathrm{d}v_\alpha}{\mathrm{d}t} = \frac{v'_e(d_\alpha, v_\alpha) - v_\alpha}{\tau} \tag{3}$$

$$v'_\alpha(d_\alpha, v_\alpha) = \frac{v_0}{2} f(d_\alpha, v_\alpha) \tag{4}$$

$$f(d_\alpha, v_\alpha) = \begin{cases} \tanh(\frac{d_\alpha}{v_\alpha+\epsilon} - 1) + \tanh(1), & \text{if } d_\alpha > 10\text{m} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where ϵ is an infinitesimally small positive number.

(i) Analyse this new model specification and explain what the behavioural interpretation of the model is.

The basis of both of these models is to adjust the vehicles velocity based upon the vehicle in-front. The new model specification specifically only accelerates if the actual distance to the vehicle in front is greater than 10m compared to the Optimal Velocity Model (OVM) which accelerates and decelerates primarily based upon an expected distance. The OVM also is controlled by a speed limit while the alternative specification continues to accelerate with no indication of a speed limit.

(ii) Come up with two specific examples (either specific modelling scenarios or specific properties of the models), where one of the two model specifications is more realistic than the other one.

- The OVM is more realistic when placed in steady state traffic i.e. when travelling in a speed limited zone where everyone accelerates until the same speed is achieved.
- The alternate specification is more realistic at modelling traffic shock-waves i.e. moving off after a temporary stoppage and the impact this has on vehicles with backwards propagation.

(iii) Compare model specifications 1 and 2; explain what the differences and similarities in model dynamics is for the following three cases:

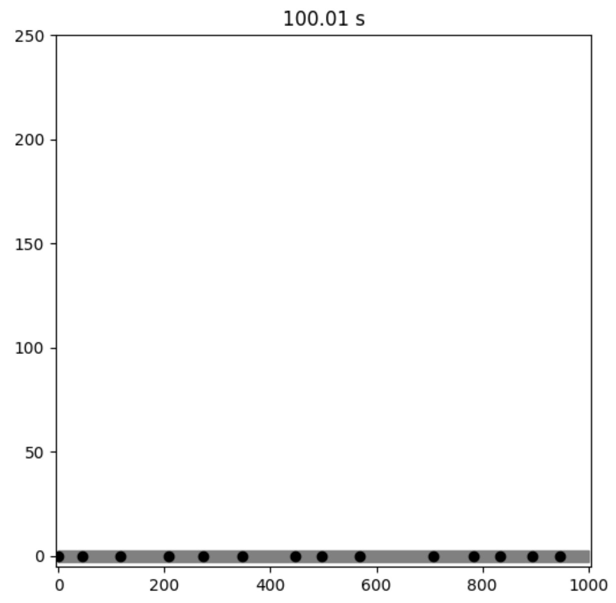
1. Steady state traffic (i.e., free flowing traffic with no particular perturbations or obstructions).

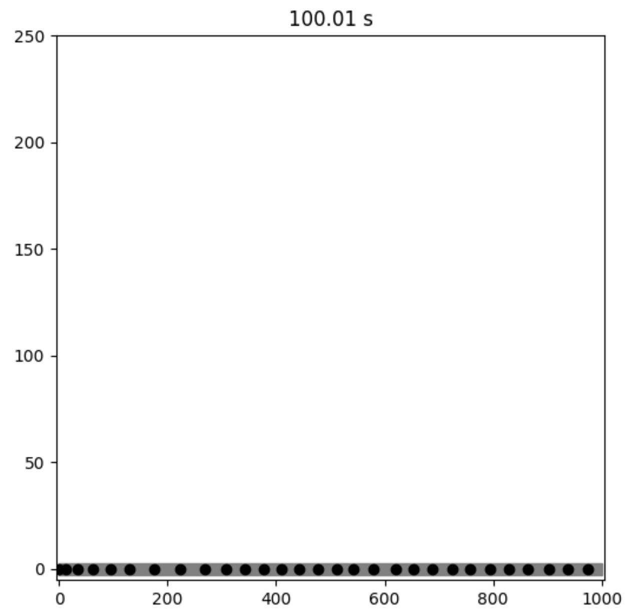
2. A queue of cars at a traffic light that has just turned green (assume that cars are spaced at 5 metres initially (net gap, bumper to bumper).

3. A sudden crash of one of the cars. The car has crashed into a stationary object and has reduced its speed to zero in approximately zero seconds.

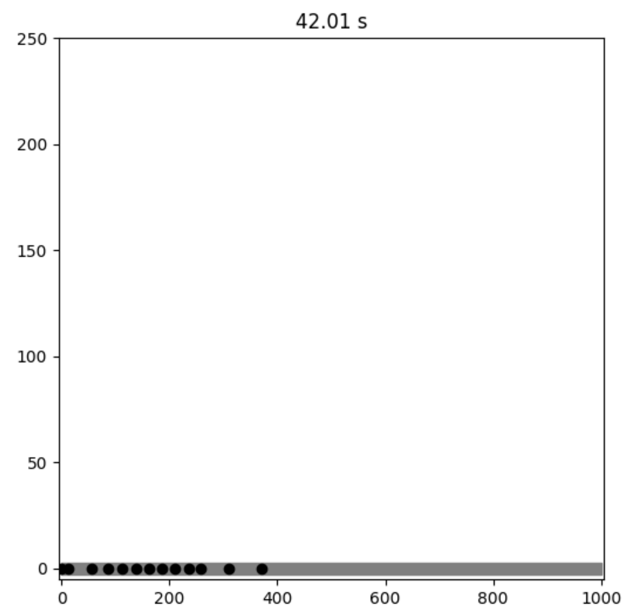
(OVM simulation is shown on the left while alternate model simulation is shown on the right)

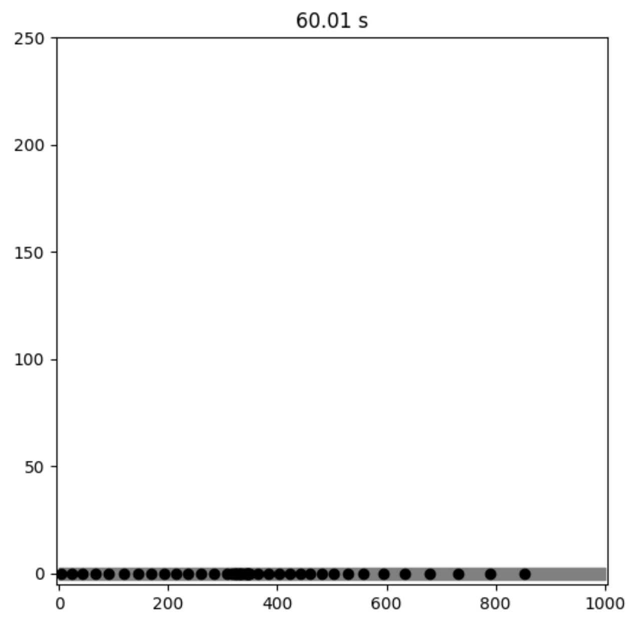
1. The OVM shows linear, evenly-spaced traffic flow that travels often at the maximum speed and only accelerates when the space ahead is available. The alternate specification on the other hand shows congested traffic often with vehicles not able to accelerate often and not achieving their maximum speed.





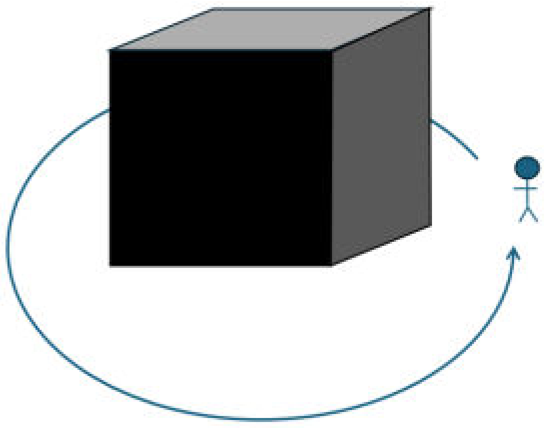
2. In the OVM, cars queue evenly while in the alternate model they do not. Therefore when the traffic light turns green in the OVM cars move off one-by-one with some gap between them and the car in front while in the alternate model specification all cars move off as soon as they can.





3. In this scenario it is understood that the cars in the both would slow down and stop as soon as they are close enough to the car in-front to be impacted. The vehicles in the OVM do this in a way that appears in terms of cautious, sensible driving i.e. slowing down well before the hazard ahead is met positionally. The alternate model on the other hand appears to do this in a less cautious way, only slowing down when absolutely essential to not cause a crash themselves.

(b) Consider the pilgrimage to Mecca where each pilgrim will walk 7 times around a central cubic building called Kaaba.



To simulate this scenario, write down a modified set of Social Force Model equations and add the following behaviours:

• The walking direction of each person should be circular, around a central point, in a counter-clockwise direction.

• If the local density around a pedestrian exceeds 6m^−2 (measured within a 5 metre radius around that person), an additional social force will act upon that pedestrian away from the central point, in order to seek out a lower density environment.

• When a pedestrian has completed the seven laps, the person will change their walking direction to be tangentially away from the centre.

General form of the social force model equations:

$$\frac{d\vec{v}_\alpha(t)}{dt} = \vec{f}_\alpha(t) + \overset{\text{Noise term}}{\vec{\xi}_\alpha(t)}$$

Where:

$$\vec{f}_\alpha(t) = \frac{1}{\underset{\text{Acceleration time}}{\tau_\alpha}}\Big(\underset{\text{Desired velocity}}{v^0_\alpha \vec{e}_\alpha} - \underset{\text{Actual velocity}}{\vec{v}_\alpha}\Big) + \overbrace{\sum_{\beta(\neq\alpha)} \vec{f}_{\alpha\beta}(t)}^{\text{Forces from all other pedestrians } \beta} + \underbrace{\sum_i \vec{f}_{\alpha i}(t)}_{\text{Forces from all boundaries.}}$$

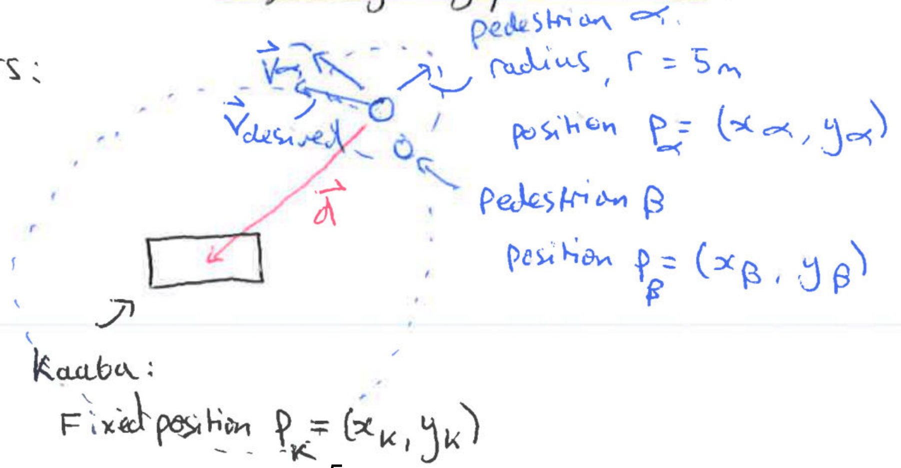
Following behaviours need to be added:

1. The walking ~~[illegible]~~ direction of each person should be circular around a central point, in a counter clockwise direction.

2. If the local density around a pedestrian exceeds $6m^{-2}$ (measured with a 5-meter radius around that person), an additional social force will act upon that pedestrian away from the central point, in order to seek out a lower density environment

3. When a pedestrian has completed the seven laps, the person will change their walking direction to be tangentially away from the centre.

Diagram of behaviours:



$\vec{V}_\alpha$
$\vec{V}_{desired}$
$\vec{d}$
pedestrian α:
radius, $r = 5m$
position $P_\alpha = (x_\alpha, y_\alpha)$
Pedestrian β
position $P_\beta = (x_\beta, y_\beta)$
Kaaba:
Fixed position $P_K = (x_K, y_K)$

For behaviours ① and ③ the desired velocity needs to be modified :

$$\vec{v}_{desired} = V^{o}_{\alpha}\, \vec{e}_{\alpha}$$

V^{o}_{α} is a constant and so isn't changed. Therefore $\vec{e}_{\alpha}$ must be changed. This is known as the pedestrian α's desired direction and is defined as follows to simulate behaviours ① and ③ :

$$\vec{e}_{\alpha} = \begin{cases} \dfrac{(d_j, -d_i)}{\|\vec{d}\|} & D_{\alpha} < D_{\text{seven Laps}, \alpha} \\ \dfrac{\vec{V}_{\alpha}}{\|\vec{V}_{\alpha}\|} & \text{Otherwise} \end{cases}$$

Where :

$$\vec{d} = (d_i, d_j) = \frac{P_K - P_{\alpha}}{\|P_K - P_{\alpha}\|}$$

And :

$$D_{\alpha} = \text{Distance covered by pedestrian (Total over simulation)} = \sum_{i=1}^{t} V^{i}_{\alpha}(t)\, dt$$

$$D_{\text{seven Laps}, \alpha} = \text{Distance to be covered for seven laps to be complete (for each pedestrian } \alpha) = 7\left(2\pi \|\vec{d}\|\right)$$

For behaviour ②, the noise term should be edited to be the following:

$$\vec{\xi}_\alpha(t) = \begin{cases} \dfrac{(-di, -dj)}{\|\vec{d}\|} & \rho^{\alpha}_{5\text{-metre}} > 6 \\ 0 & \text{otherwise} \end{cases}$$

Where $\rho^{\alpha}_{5\text{-metre}}$ is the local density in a 5-metre radius around pedestrian α. This is calculated using the following equation:

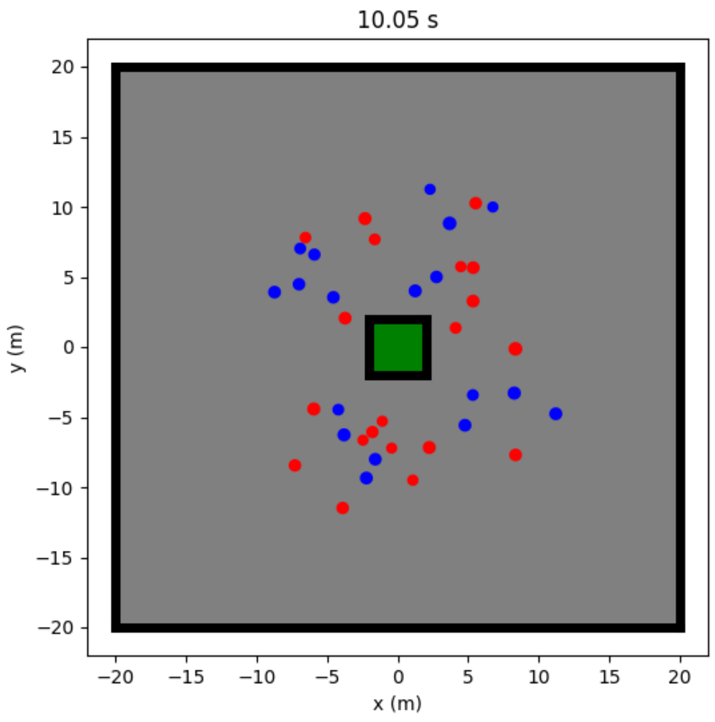
$$\rho^{\alpha}_{5\text{-metre}} = \sum_{(\beta \neq \alpha)} \rho_{\alpha\beta}$$

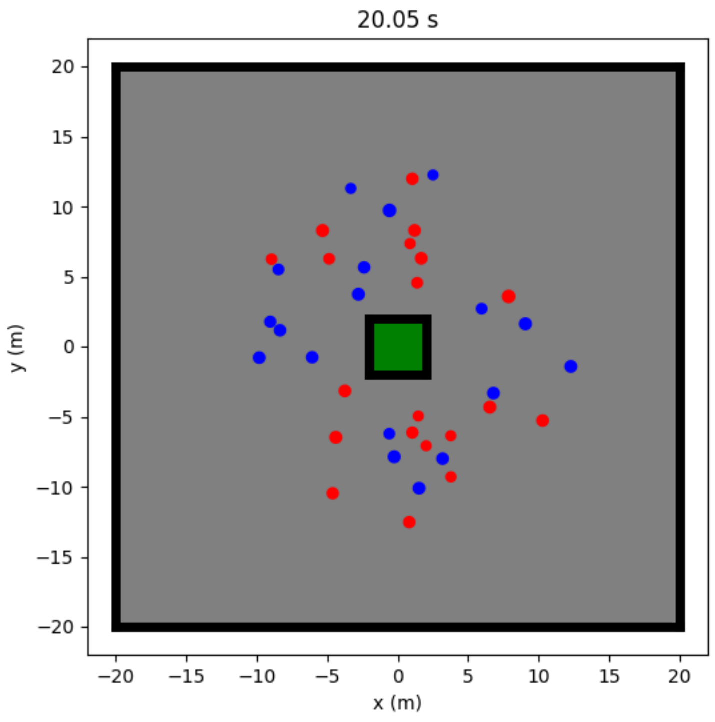
Where:

$$\rho_{\alpha\beta} = \begin{cases} 1 & \sqrt{(x_\alpha - x_\beta)^2 + (y_\alpha - y_\beta)^2} < 5 \\ 0 & \text{otherwise} \end{cases}$$

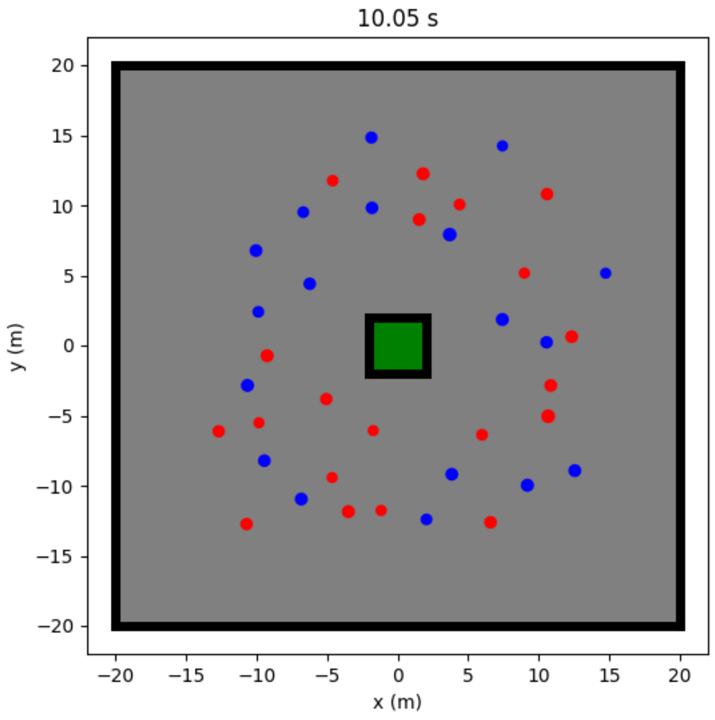
An example of an implementation of behaviors 1 and 3 using python code provided is shown below. Unfortunately condition 2 was not able to be applied due to time-constraints.

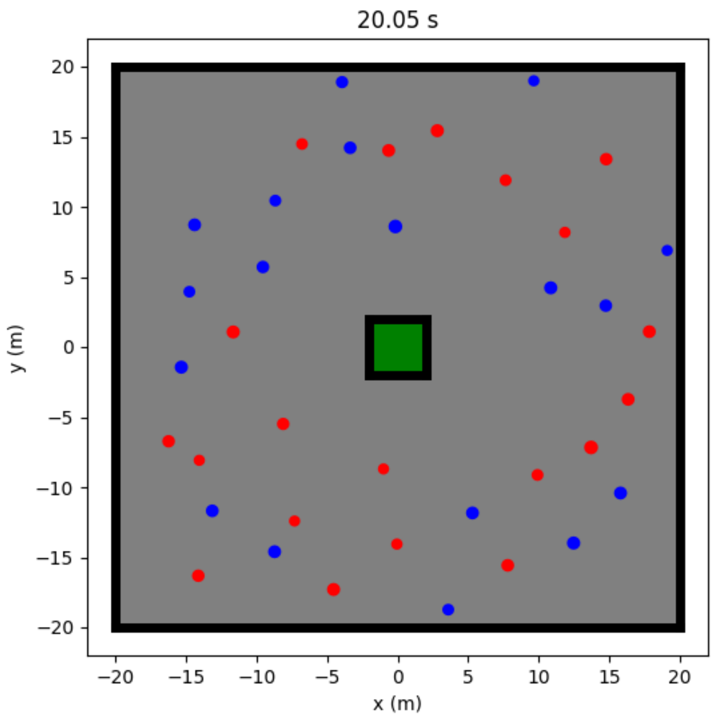
Pictures 1 & 2: The walking direction of each person should be circular, around a central point, in a counter-clockwise direction (i.e. condition 1):

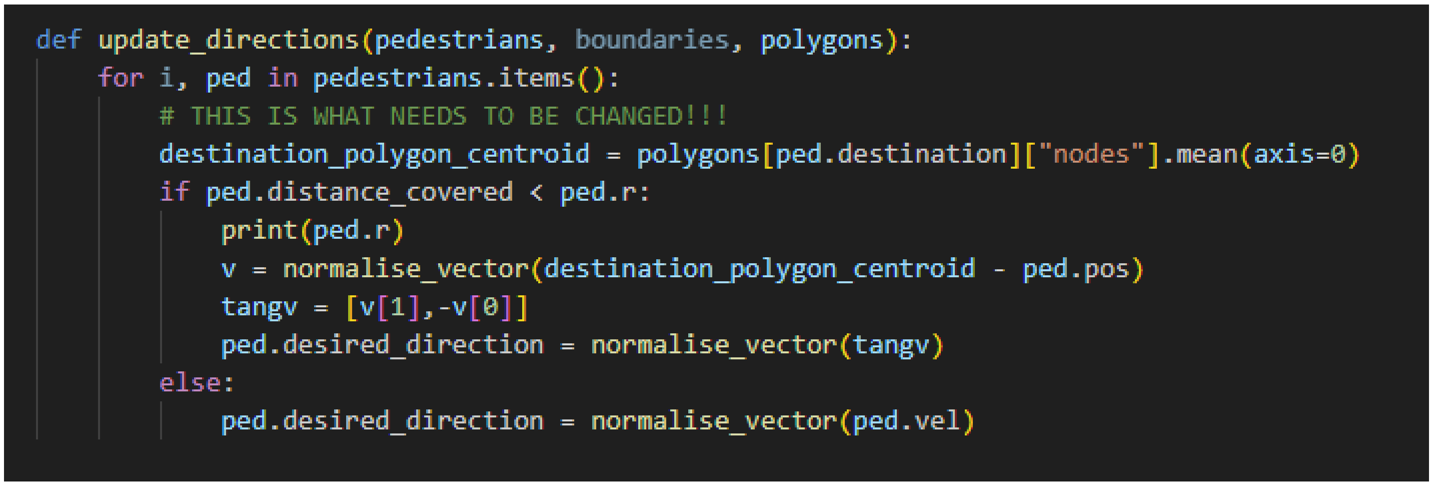




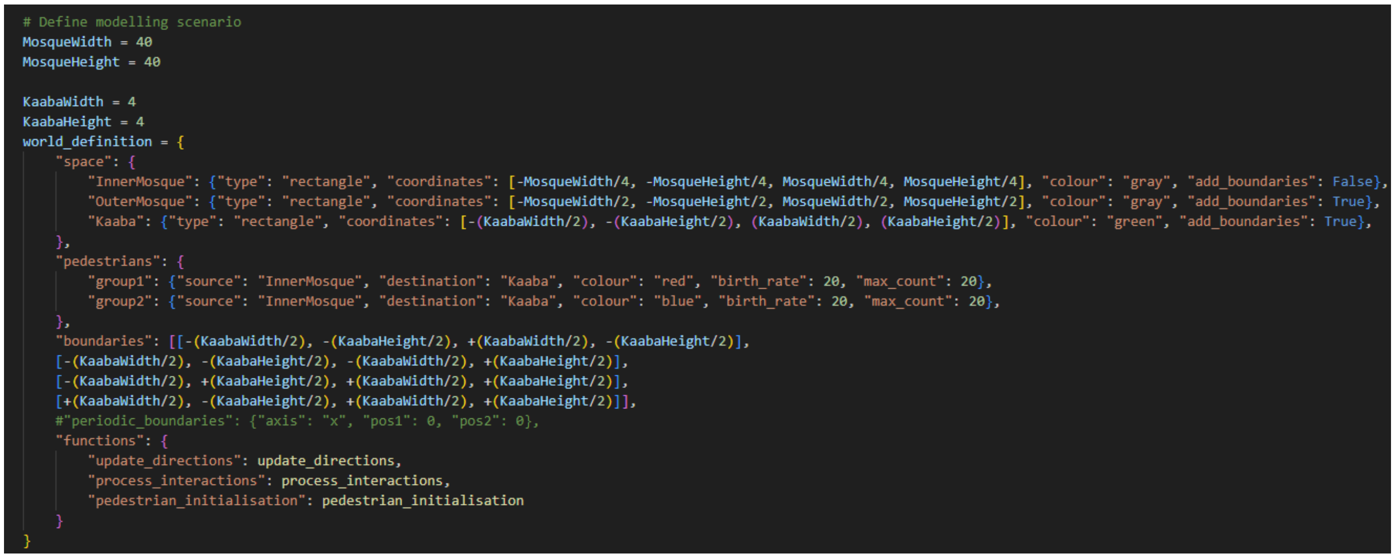
Pictures 3 & 4: when a pedestrian has completed the seven laps, the person will change their walking direction to be tangentially away from the centre (i.e. condition 3):





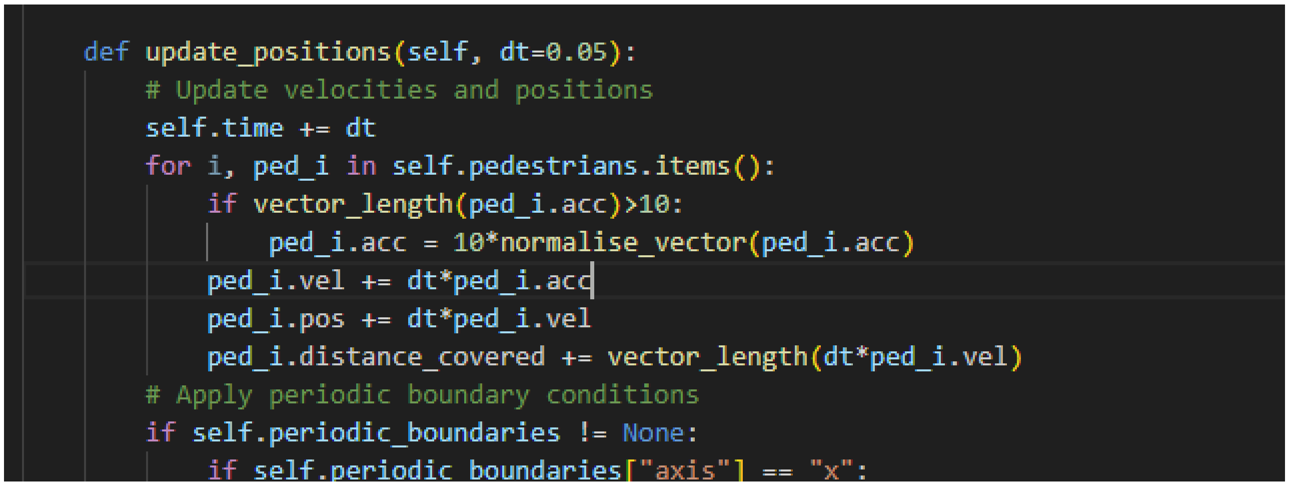


```python
def update_directions(pedestrians, boundaries, polygons):
    for i, ped in pedestrians.items():
        # THIS IS WHAT NEEDS TO BE CHANGED!!!
        destination_polygon_centroid = polygons[ped.destination]["nodes"].mean(axis=0)
        if ped.distance_covered < ped.r:
            print(ped.r)
            v = normalise_vector(destination_polygon_centroid - ped.pos)
            tangv = [v[1],-v[0]]
            ped.desired_direction = normalise_vector(tangv)
        else:
            ped.desired_direction = normalise_vector(ped.vel)
```



```python
# Define modelling scenario
MosqueWidth = 40
MosqueHeight = 40

KaabaWidth = 4
KaabaHeight = 4
world_definition = {
    "space": {
        "InnerMosque": {"type": "rectangle", "coordinates": [-MosqueWidth/4, -MosqueHeight/4, MosqueWidth/4, MosqueHeight/4], "colour": "gray", "add_boundaries": False},
        "OuterMosque": {"type": "rectangle", "coordinates": [-MosqueWidth/2, -MosqueHeight/2, MosqueWidth/2, MosqueHeight/2], "colour": "gray", "add_boundaries": True},
        "Kaaba": {"type": "rectangle", "coordinates": [-(KaabaWidth/2), -(KaabaHeight/2), (KaabaWidth/2), (KaabaHeight/2)], "colour": "green", "add_boundaries": True},
    },
    "pedestrians": {
        "group1": {"source": "InnerMosque", "destination": "Kaaba", "colour": "red", "birth_rate": 20, "max_count": 20},
        "group2": {"source": "InnerMosque", "destination": "Kaaba", "colour": "blue", "birth_rate": 20, "max_count": 20},
    },
    "boundaries": [[-(KaabaWidth/2), -(KaabaHeight/2), +(KaabaWidth/2), -(KaabaHeight/2)],
    [-(KaabaWidth/2), -(KaabaHeight/2), -(KaabaWidth/2), +(KaabaHeight/2)],
    [-(KaabaWidth/2), +(KaabaHeight/2), +(KaabaWidth/2), +(KaabaHeight/2)],
    [+(KaabaWidth/2), -(KaabaHeight/2), +(KaabaWidth/2), +(KaabaHeight/2)]],
    #"periodic_boundaries": {"axis": "x", "pos1": 0, "pos2": 0},
    "functions": {
        "update_directions": update_directions,
        "process_interactions": process_interactions,
        "pedestrian_initialisation": pedestrian_initialisation
    }
}
```



```python
    def update_positions(self, dt=0.05):
        # Update velocities and positions
        self.time += dt
        for i, ped_i in self.pedestrians.items():
            if vector_length(ped_i.acc)>10:
                ped_i.acc = 10*normalise_vector(ped_i.acc)
            ped_i.vel += dt*ped_i.acc
            ped_i.pos += dt*ped_i.vel
            ped_i.distance_covered += vector_length(dt*ped_i.vel)
        # Apply periodic boundary conditions
        if self.periodic_boundaries != None:
            if self.periodic_boundaries["axis"] == "x":
```