StationSim GCS

Keiran Suchak

January 3, 2021

1 ODD Protocol

ODD (Overview, Design Concepts, Details) is a protocol for outlining and documenting agent-based models [1, 2, 3]. The aim of such a protocol is to allow researchers to share and reproduce their models via a standardised format. The protocol consists of three components, each of which cover a number of different topics:

1. Overview providing an overview of the model, touching upon each of the following topics outlined in Section 1.1.
2. Design Concepts: detailing which of the following design concepts are relevant to the model (detailed in Section 1.2).
3. Details: elaborating on the internal mechanics of the model, which will cover each of the topics outlined in Section 1.3.

1.1 Overview

The Overview section of ODD aims to provide a cursory overview of the model that is being documented. This includes subsections on each of the following topics:

1. Purpose and patterns: What can a user expect the model to do/achieve, and what patterns might the model be fit to simulation [4]?
2. Entities, state variables and scales: What types of agents and environments are we looking to model, what variables are we using to represent them, and what spatial and temporal scales are we working at?
3. Process overview and scheduling: What processes govern the interactions in the model, and in what order to they occur?

This information should enable others to develop and outline of the what the model looks like at a functional level.

1.2 Design Concepts

When designing an agent-based model, there are a number of design concepts that are often incorporated. The following is a non-exhaustive list of the design concepts that may be involved in an agent-based model:

1. Basic principles
2. Emergence
3. Adaptation
4. Objectives
5. Learning
6. Prediction
7. Sensing
8. Interaction
9. Stochasticity
10. Collectives
11. Observation

These design concepts are detailed in greater depth in [2].

1.3 Details

The Details section of ODD aims to build upon the above sections, providing the specifics required to build a fully functioning model. This takes the form of the following topics:

1. Initialisation: What information is required to start the model with respect to defining the initial model state and calibrating parameter values?
2. Input data: What information does the model require as it runs?
3. Submodels: How do the individual processes governing model behaviour work?

2 ODD for StationSim GCS

2.1 Overview

2.1.1 Purpose and patterns

StationSim GCS is an updated version of the StationSim model. The original StationSim aimed to simulate the motion of pedestrians across a hypothetical rectangular station with 3 entrances on one side and 2 exits on the opposite side as shown in Figure 1. The new StationSim GCS also aims to simulation the motion of pedestrians across a station; however, in this case the model is based on the real-world example of Grand Central Station in New York, focusing specifically on the concourse area highlighted in Figure 2. This is reflected in the simulation environment shown in Figure 3. The environment consists of X gates which act simultaneously as both entrances and exits. Each pedestrian in the simulation is assigned an entrance and an exit and, upon entering the environment, seeks to move as directly as possible towards their assigned exit without colliding with other pedestrians. Where collisions are more likely to occur (e.g. close to entrances/exits and around solid obstacles), we typically observe crowding as population densities increase.

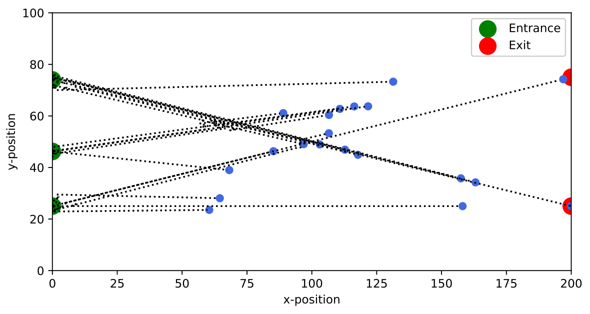


Figure 1: Layout of environment in original StationSim model.

2.1.2 Entities, state variables and scales

The StationSim GCS model is made up of 4 different types of entities:

1. Agents,
2. The environment,
3. Gates around the edge of the environment, and
4. Obstacles in the environment.

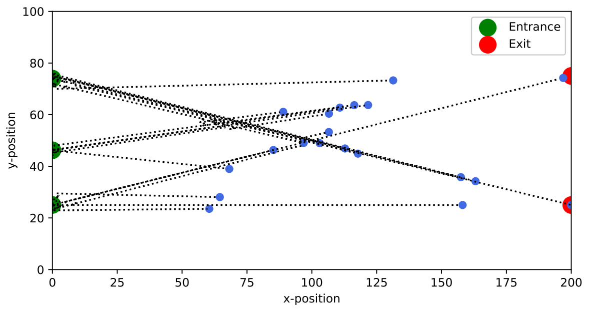


Figure 2: Layout of Grand Central Station concourse.

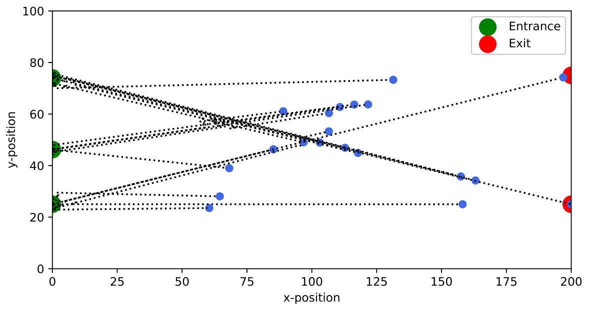


Figure 3: Layout of environment in StationSim GCS model.

These entities aim to simulate the scenario outlined in Section 2.1. The agents in this model represent pedestrians; these are portrayed as two-dimensional circular entities with finite radius. The variables pertaining to these agents can be found in Table 1. The environment in this model represents the concourse of Grand Central Station in New York; this is portrayed as two-dimensional continuous space bounded by rectangular walls within which agents may move. The model is designed such that the left-hand side of the environment represents the South side of the concourse, the right-hand side the North side, the top side the West side and the bottom side the East side. The variables pertaining to the environment entity can be found in Table 2.

Along the edge of these boundaries are located gates: one gate is located on the South side, five gates are located on the North side, two gates are located on the West side and two gates are located on the East side. The gates are points along the boundary of the environment at which agents may either enter or exit. These have specific fixed x-y coordinates. Upon initialisation, each agent is provided with a start gate and end gate, from which it draws its initial location and target destination; in defining its target destination, the agent introduces some random noise to the x-y coordinate

Variable Name	Description
`location`	Agent's x-y coordinates in 2-dimensional continuous space; bounded by the height and width of the environment
`status`	Agent's status; 0 indicates agent has not started, 1 indicates agent is active, 2 indicates agent has finished
`size`	Radius of agent's circular body
`speed`	Agent's speed; indicative of the distance covered by an agent in a single time-step
`unique_id`	Unique numerical identifier for a specific agent in a model
`gate_in`	Number of the gate through which of the gates the agent enters the environment ($0 \leq n \leq 9$)
`gate_out`	Number of the gate through which of the gates the agent exits the environment ($0 \leq n \leq 9$)
`loc_desire`	x-y coordinate of the agent's target destination; defined by taking the x-y coordinates of `gate_out` and adding some uniformly distributed random noise

Table 1: Table of state variables pertaining to gate entities.

Variable Name	Description
`height`	Environment's height
`width`	Environment's width
`gates_in`	Number of gates through which agents can enter the environment
`gates_out`	Number of gates through which agents can exit the environment

Table 2: Table of state variables pertaining to the environment entity.

Variable Name	Description
`location`	Gate's x-y coordinates; restricted to one of ten distinct locations on the boundary of the environment

Table 3: Table of state variables pertaining to gate entities.

Variable Name	Description
`location`	Obstacle's x-y coordinates
`size`	Radius of obstacle's circular body
`speed`	Speed of agent characterising the obstacle; fixed value of 0

Table 4: Table of state variables pertaining to the obstacle entity.

in order to emulate the non-zero width of the gate. The variables pertaining to the gate entities can be found in Table 3.

The environment also contains a single obstacle which represents a clock. As shown in Figure 2, this lies in the centre of the concourse. From a model architecture perspective, this obstacle is treated as a stationary agent; other agents therefore treat it as they would any other agent and make efforts to avoid colliding with it. The variables pertaining to the obstacle entity can be found in Table 4.

Of the variables detailed for each of the entities in Tables 1, 2, 3 and 4, the majority are set to fixed values upon model initialisation. The variables that change as the model runs are the `location` and `status` variables pertaining to the agent entities.

2.1.3 Process overview and scheduling

The overall model process is relatively simple; it is outlined in Figure 4. This process consists of 4 components:

1. Set up the station.
2. Initialise agents.
3. Step the model forward.
4. Check whether the model should be deactivated.

The process of setting up the station involves defining the state of the non-agent entities in the model, i.e. the model boundaries, the gates and the central clock obstacle, as outlined in Tables 2, 3 and 4 respectively.

The process of initialising the agents in the model involves allocating their entrance and exit gates (along with their target destination), their speed and the time at which they will be activated (i.e. when they will attempt to enter the system).

The processes of stepping the model forward and checking for model deactivation are run iteratively whilst the model is still deemed active. After having stepped the model forward, it is checked whether the model should remain active; if so, the model stepping process is repeated, else the model run is completed.

The process of stepping the model forward involves the following set of subprocesses:

1. Checking whether inactive agents should be activated.
2. Identifying all of the potential collisions that may occur between agents and between agents and parts of the environment (i.e. the model boundaries and the clock obstacle).
3. Moving each of the active agents for a specific time interval based on the when the soonest collision will occur.
4. Where necessary, instruct agents to engage a sidestepping mechanic to avoid obstacles.

Each agent is stepped sequentially as part of the model stepping process; the agent stepping process involves moving from its present location towards its target destination in a linear fashion. The movement vector is the product of the agents velocity vector and the time-step, which is dependent on the time interval to the next expected collision. As a consequence, although the agents are stepped sequentially and, no collisions or interactions are missed.

Each of the above processes are outlined in more detail in Section 2.3.3.

2.2 Design Concepts

Of the design concepts outlined in [2], the following are considered relevant to this model:

1. Emergence
2. Adaptation
3. Prediction
4. Sensing
5. Interaction
6. Stochasticity
7. Observation

As outlined in Section 2.1.3, in each time-step agents engage in collision avoidance (both with each other and with walls and obstacles). This is achieved by *predicting* the paths that agents would take if they were to continue moving towards their target destinations; this is made possible through the use of a k-d tree which emulates a form of *sensing* whereby each agent is aware of the position of other agents who are likely to collide with them. In cases where collisions would occur, the agent paths are *adapted*. Such adaptations can be considered an indirect form of *interaction* between an agent and other agents (as well as stationary objects in the environment).

Stochasticity is incorporated in the model in number of different ways. Upon initialisation, agents are randomly allocated an entrance gate and an exit gate; entrance gates are sampled from a uniform discrete distribution, and exit gates are sampled from a uniform discrete distribution excluding the gate through which the agent is entering. In cases where pedestrians are predicted to collide, part of the avoidance process involves the addition of normally distributed random noise to the agent's movement vector. Finally the time at which each agent enters the model is sampled from an exponential distribution.

Whilst the model is running, *observation* is undertaken by collecting information on the positions of each agent at each time-step for comparison with pseudo-truth data. In scenarios where pedestrian density is sufficiently high, we observe the *emergence* of crowding behaviour.

2.3 Details

2.3.1 Initialisation

2.3.2 Input data

Data regarding pedestrian movements are used to calibrate the model prior to running. The data used for this process are trajectory data sourced from [5]. The manner in which these data are used is outlined in Section 2.3.1. Beyond this, not external data are used to run the model.

2.3.3 Submodels

This section of the ODD outlines the specific details of how each of the submodels are implemented. In each case, these details are accompanied by a flow diagram to illustrate how each process works.

StationSim Setup The initial setup of StationSim involves setting up three entities: the model boundaries, the gates around the edges of the model and environment and the clock obstacle at the centre of the environment. The environment is defined by its dimensions which are outlined in Section 2.1.2. Based on these values, an array of boundaries are defined.

Given each of these boundaries, the gates are defined; these are defined as a series of 11 sets of coordinates located on the edges of the model environment. Each of the gates are also allocated a width which reflects their width in the real station. Finally, the clock is defined; the clock is initialised as a stationary agent with size and location given in Section 2.1.2. These process occur sequentially as outlined in Figure 5.

In addition to setting up each of these entities, variables outlining the distribution of agent speeds are also initialised.

Agent Initialisation Following the process of setting up the model station, agents are initialised to characterise a pedestrian population in the model (as shown in Figure 4). For each agent, the initialisation process involves a number of steps, the first of which is to define the agent's status as inactive (i.e. not started); it is worth noting that an agent's status can take one of three values — inactive (not started), active and inactive (finished).

Agents are then allocated an entrance gate — this process is described in Section 2.3.3. The agent's exit gate is then chosen — the process by which this happens is outlined in Section 2.3.3. Given an exit gate, the agent's target destination location and speed are allocated as per the processes outlined in Section 2.3.3 and 2.3.3. Finally, the agent is allocated an activation time, i.e. a model time-step at which is should change its status from inactive to active and proceed towards its target destination. The time at which the agent should be activated is governed by the model birth-rate:

$$t_a = i \times \frac{25}{\lambda} \tag{1}$$

where t_a is the agent's activation time, i is the agent ID and λ is the model birth-rate.

Agent Entrance Allocation The gate through which an agent enters is chosen at random, with each of the 11 gates carrying equal probability. The entrance gate value for an agent can, therefore, take any integer value between 0 and 10 (inclusive).

Agent Exit Allocation The process of choosing an exit gate is slightly more complicated; it is considered unlikely that pedestrians will exit through a gate on the same edge of the environment as the gate through which they enter, and as such the agent's exit gate is chosen at random from the gates on the sides of the environment other than the side on which the agent's entrance gate resides (again, with each potential exit gate carrying equal probability). In practice, this means that if an agent is allocated an entrance gate on the North boundary of the model environment, then its exit will be chosen at random from the gates residing on the East, South and West boundaries. This process is outlined in Figure 8.

Allocating Agent Location based on Gate Number

Allocating Agent Speed

Model Step

Checking for Agent Activation

Checking for Potential Collisions

Agent Step

Agent Movement

Agent Wiggle

Checking for Agent Deactivation

Checking for Model Deactivation

References

[1] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K Heinz, Geir Huse, et al. A standard protocol for describing individual-based and agent-based models. *Ecological modelling*, 198(1-2):115–126, 2006.

[2] Volker Grimm, Uta Berger, Donald L DeAngelis, J Gary Polhill, Jarl Giske, and Steven F Railsback. The odd protocol: a review and first update. *Ecological modelling*, 221(23):2760–2768, 2010.

[3] Volker Grimm, Steven F Railsback, Christian E Vincenot, Uta Berger, Cara Gallagher, Donald L DeAngelis, Bruce Edmonds, Jiaqi Ge, Jarl Giske, Juergen Groeneveld, et al. The odd protocol for describing agent-based and other simulation models: A second update to improve clarity, replication, and structural realism. *Journal of Artificial Societies and Social Simulation*, 23(2), 2020.

[4] Volker Grimm and Steven F Railsback. Pattern-oriented modelling: a 'multi-scope'for predictive systems ecology. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1586):298–310, 2012.

[5] Bolei Zhou, Xiaogang Wang, and Xiaoou Tang. Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2871–2878. IEEE, 2012.

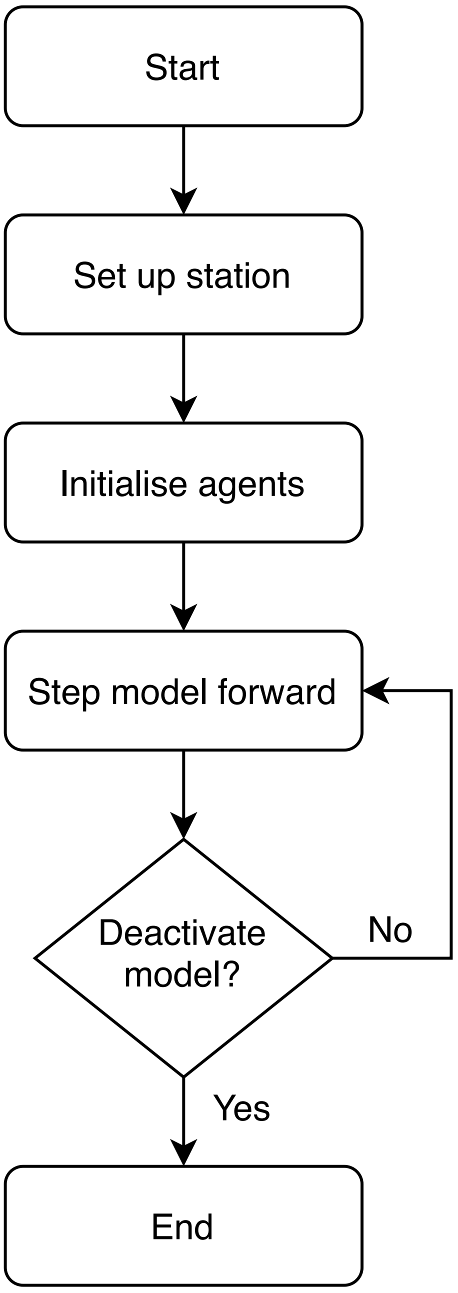


Figure 4: Flow diagram of StationSim GCS

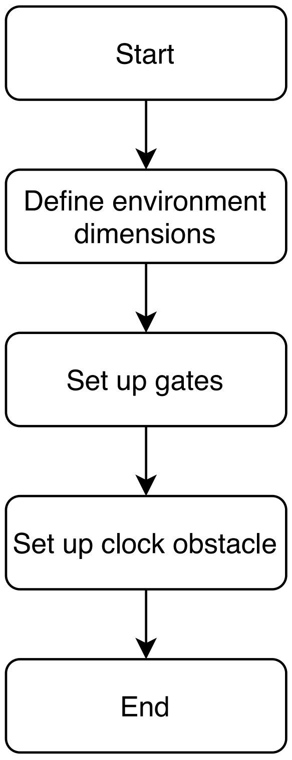


Figure 5: Flow diagram of process to set up StationSim GCS

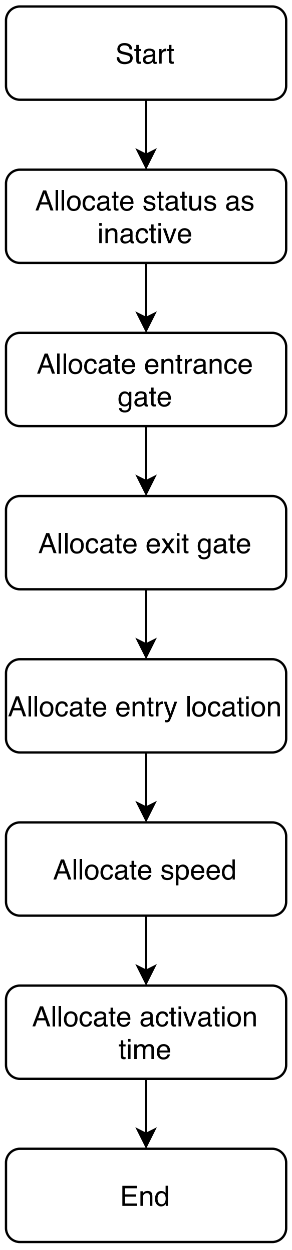


Figure 6: Flow diagram of process by which agents are initialised

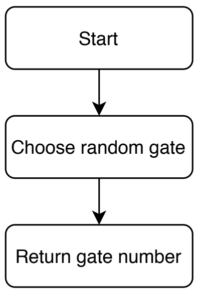


Figure 7: Flow diagram of process by which agents' entrance gates are selected

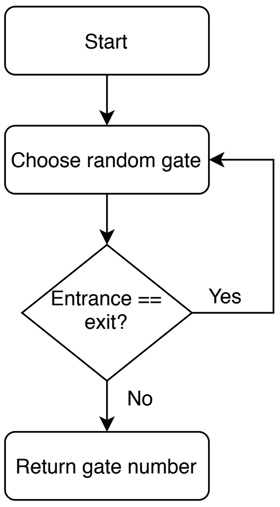


Figure 8: Flow diagram of process by which agents' exits gates are selected

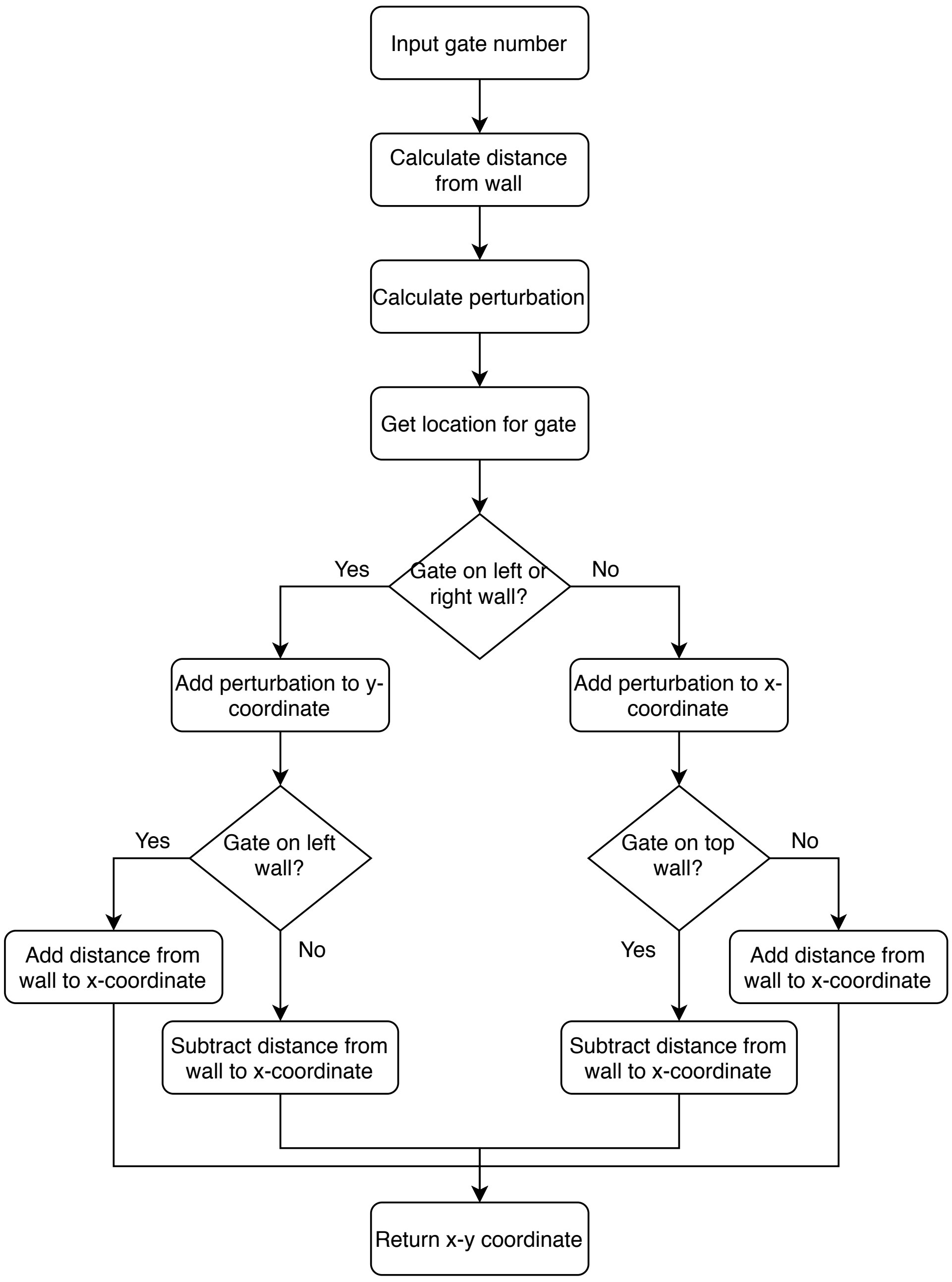

Input gate number
Calculate distance from wall
Calculate perturbation
Get location for gate
Gate on left or right wall?
Yes
No
Add perturbation to y-coordinate
Add perturbation to x-coordinate
Gate on left wall?
Yes
No
Gate on top wall?
No
Yes
Add distance from wall to x-coordinate
Subtract distance from wall to x-coordinate
Subtract distance from wall to x-coordinate
Add distance from wall to x-coordinate
Return x-y coordinate

Figure 9: Flow diagram of process by which an agent's location is allocated given a specific gate number; used to allocate initial and final locations for agents

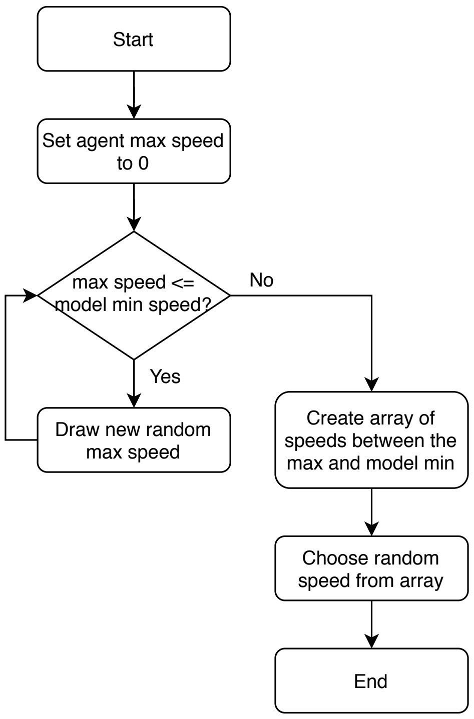
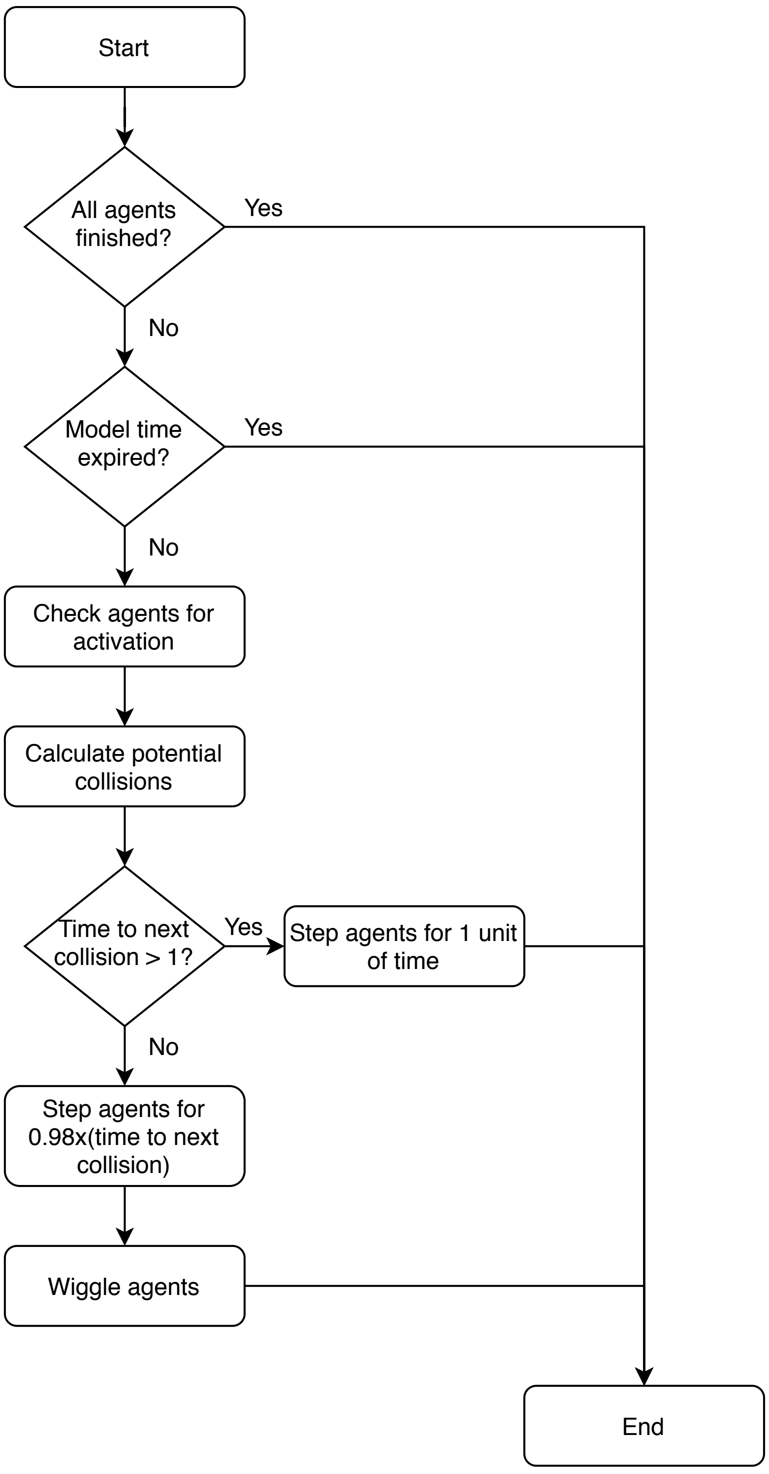

Start
Set agent max speed to 0
max speed <= model min speed?
No
Yes
Draw new random max speed
Create array of speeds between the max and model min
Choose random speed from array
End

Figure 10: Flow diagram of process by which an agent's speed is allocated



Start
All agents finished?
Yes
No
Model time expired?
Yes
No
Check agents for activation
Calculate potential collisions
Time to next collision > 1?
Yes
Step agents for 1 unit of time
No
Step agents for 0.98x(time to next collision)
Wiggle agents
End

Figure 11: Flow diagram of how the model step works

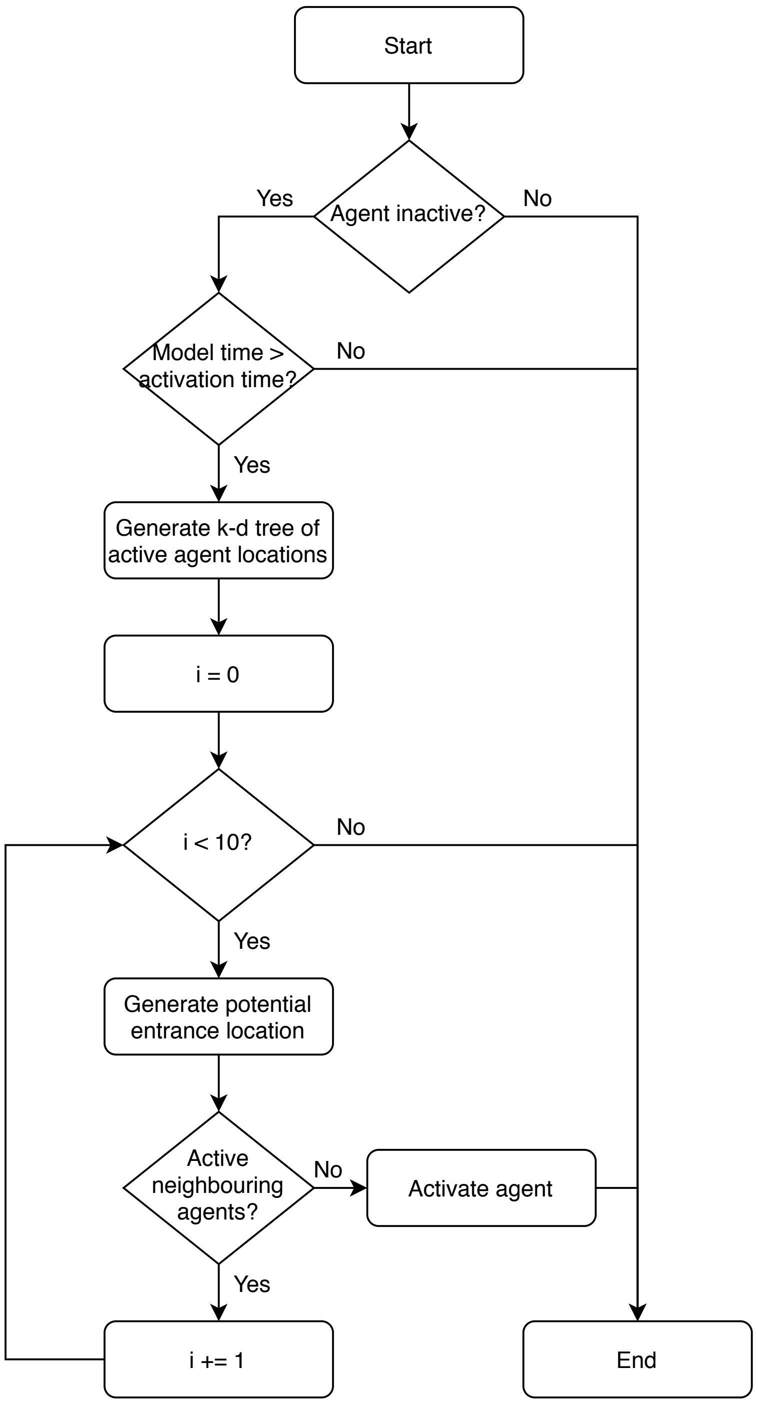

Start
Agent inactive?
Yes
No
Model time > activation time?
No
Yes
Generate k-d tree of active agent locations
i = 0
i < 10?
No
Yes
Generate potential entrance location
Active neighbouring agents?
No
Activate agent
Yes
i += 1
End

Figure 12: Flow diagram of process to check whether an agent should be activated

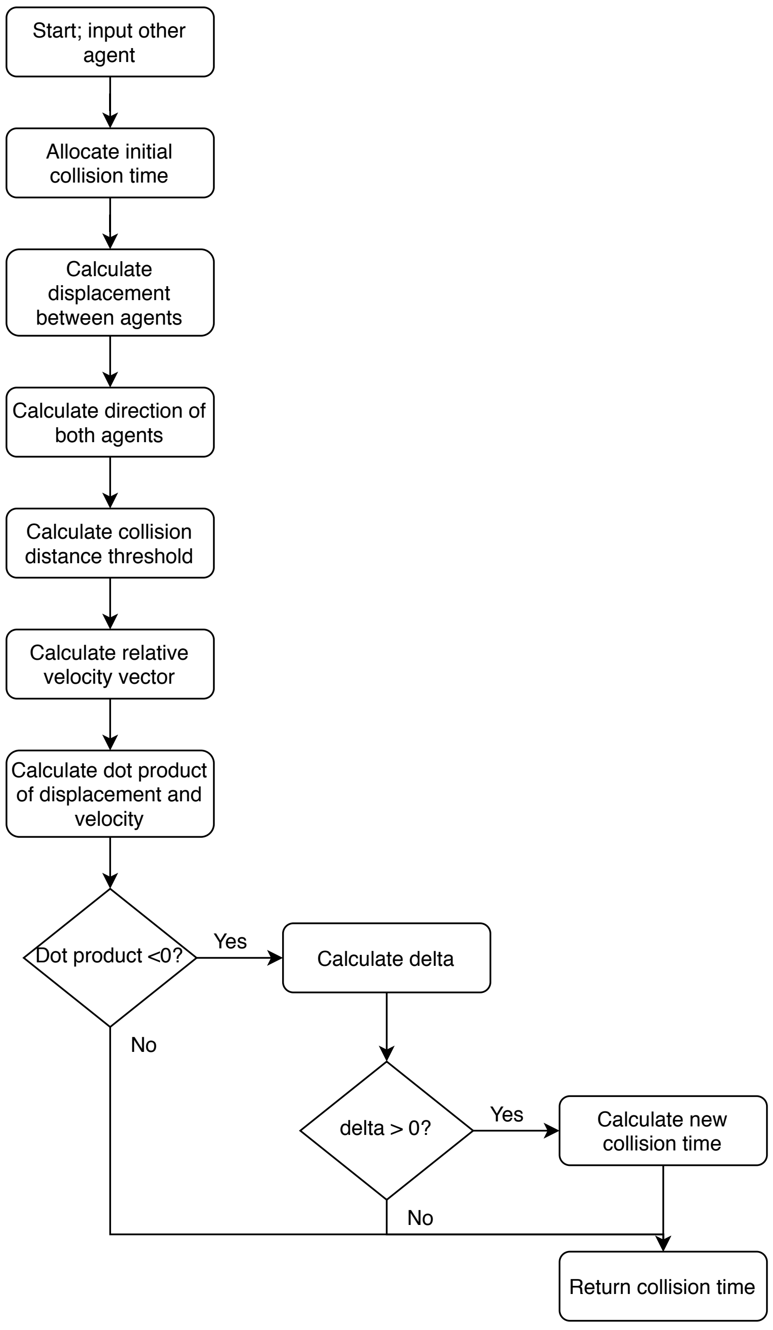

Start; input other agent
Allocate initial collision time
Calculate displacement between agents
Calculate direction of both agents
Calculate collision distance threshold
Calculate relative velocity vector
Calculate dot product of displacement and velocity
Dot product <0?
Yes
Calculate delta
No
delta > 0?
Yes
Calculate new collision time
No
Return collision time

Figure 13: Flow diagram of how the time to the next collision between two agents is calculated

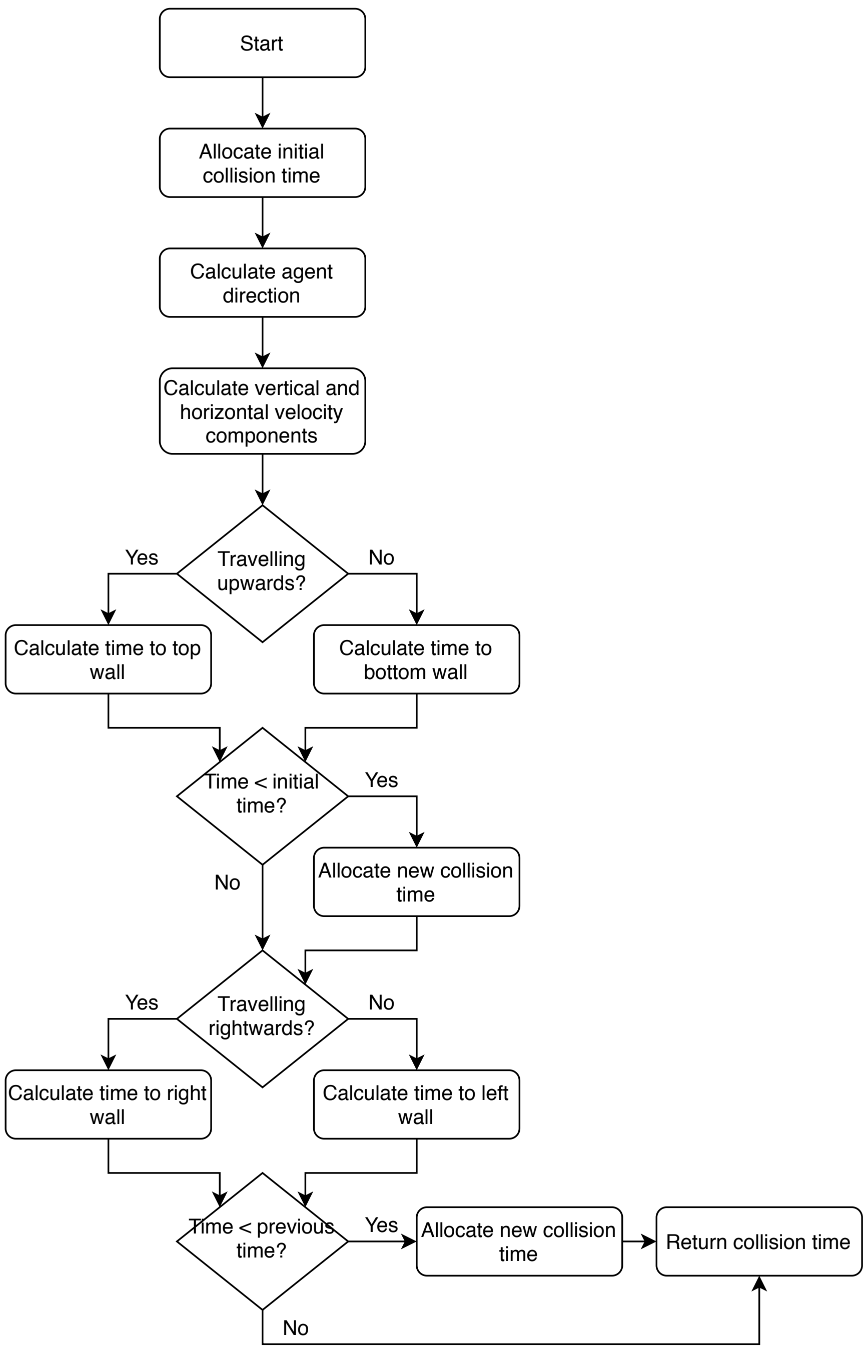

Start
Allocate initial collision time
Calculate agent direction
Calculate vertical and horizontal velocity components
Travelling upwards?
Yes
No
Calculate time to top wall
Calculate time to bottom wall
Time < initial time?
Yes
No
Allocate new collision time
Travelling rightwards?
Yes
No
Calculate time to right wall
Calculate time to left wall
Time < previous time?
Yes
No
Allocate new collision time
Return collision time

Figure 14: Flow diagram of how the time to the next collision between two an agent and a wall is calculated

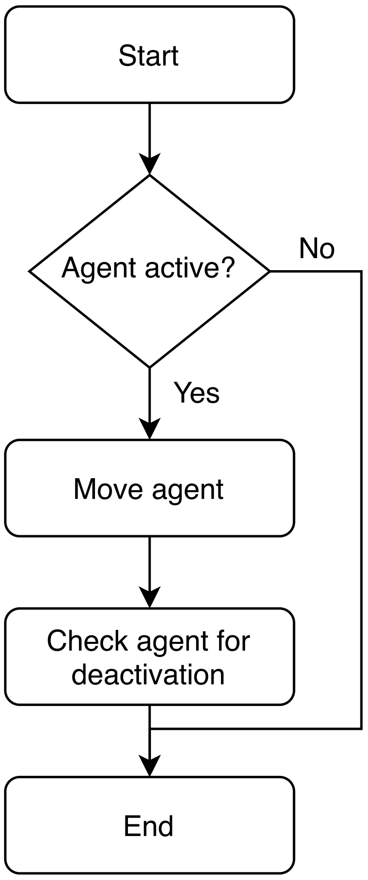


Figure 15: Flow diagram of how the agent step works

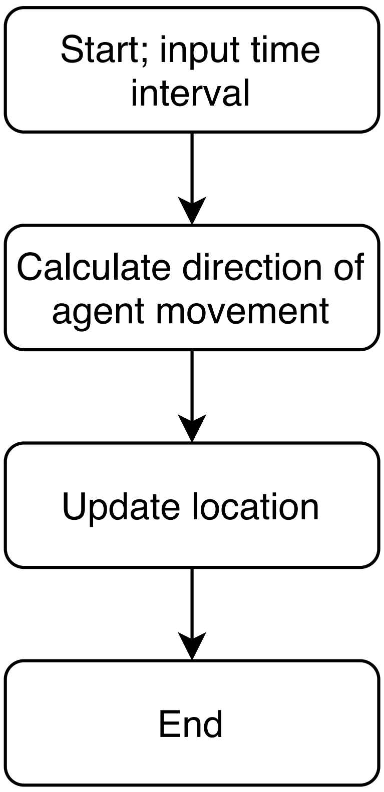

Start; input time interval
Calculate direction of agent movement
Update location
End

Figure 16: Flow diagram of how agent movements are calculated

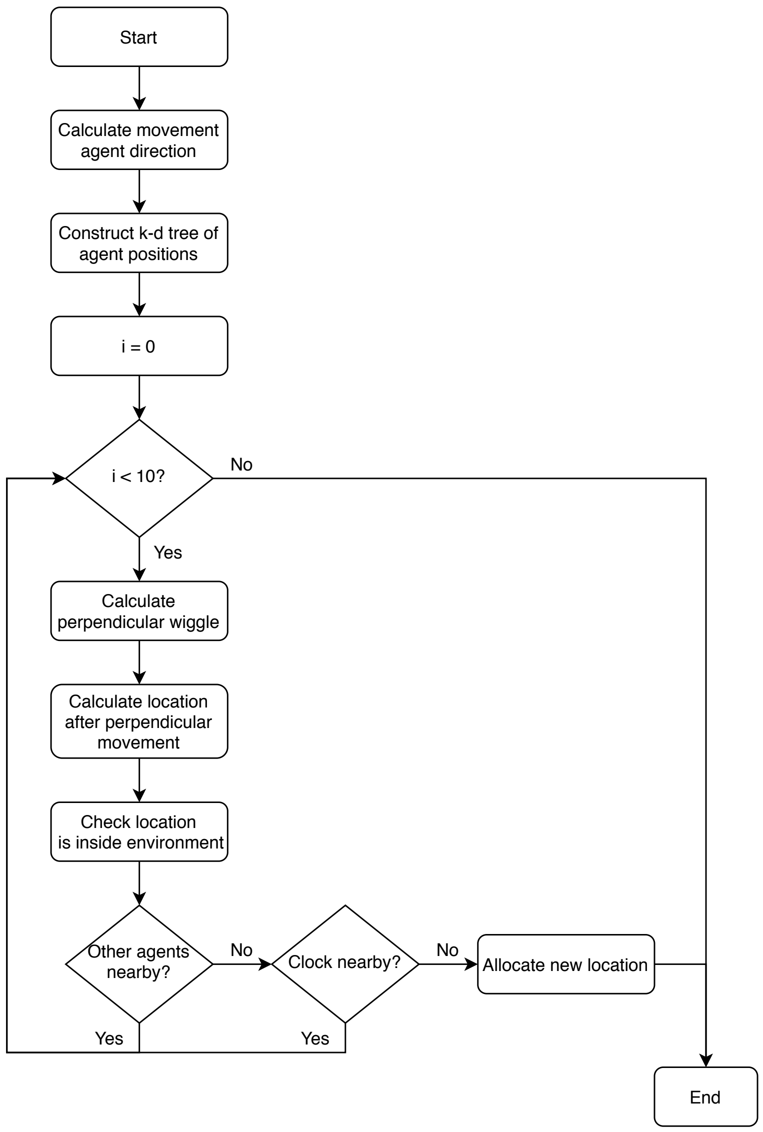

Start
Calculate movement agent direction
Construct k-d tree of agent positions
i = 0
i < 10?
No
Yes
Calculate perpendicular wiggle
Calculate location after perpendicular movement
Check location is inside environment
Other agents nearby?
No
Yes
Clock nearby?
No
Yes
Allocate new location
End

Figure 17: Flow diagram of how agent wiggles are carried out

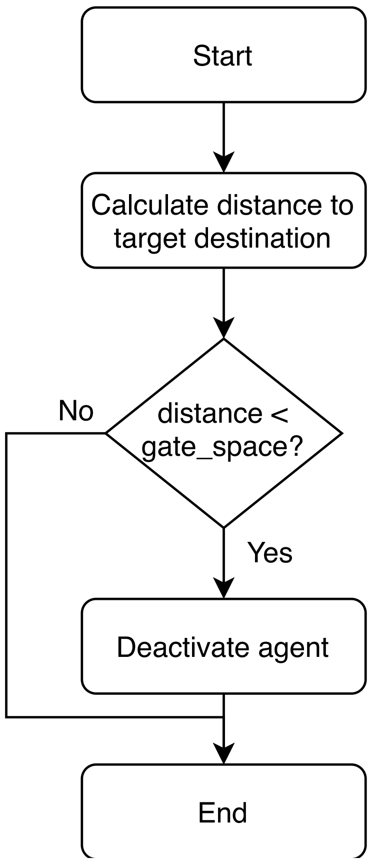


Figure 18: Flow diagram of process to check whether an agent should be deactivated